Politecnico di Torino

Politecnico di Torino

Master's Degree in

INGEGNERIA INFORMATICA (COMPUTER ENGINEERING)

a.y. 2024/2025

Graduation Session October 2025

Resource Allocation Techniques for Planning the Teaching Timetable

Supervisors:

Renato Ferrero

Sophie Fosson

Candidate:

Paolo Cagliero

Summary

The goal of this thesis is to develop a tool that supports the creation of the course timetable for a university. The timetable generation problem can be modeled as a mixed-integer optimization problem, characterized by large numerical complexity. Furthermore, in universities with a large number of Students and Teachings, the problem size increases considerably, making it even more challenging.

As a case study, I consider the Teachings of the Degree Courses related to the Computer, Cinema, and Mechatronic Engineering (ICM) and Electronic, Telecommunication, and Physical Engineering (ETF) Colleges of the Politecnico di Torino.

The timetable must satisfy many constraints. Some constraints are related to the students, e.g., mandatory Teachings cannot overlap, there cannot be too many consecutive lecture hours or too many empty Slots between lectures. Other constraints are related to the Teachers, e.g. Teachers can specify Slots in which they cannot have any lecture, and to the single Teachings, e.g. the lectures of some Teachings should be allocated in two consecutive Slots at least once per Week.

Among these constraints, I identify hard constraints, which must be respected, and soft constraints, which, if not respected, can introduce penalties in the optimization function.

Two theses were conducted on this topic, in which the timetable allocation was modeled as an Integer Linear Programming (ILP) problem and solved using the solver CPLEX. This thesis, also based on ILP, explores a novel methodology, which consists of representing the timetable as a two-dimensional matrix with the lecture Slots on the columns and the Teachings on the rows. Each element of the matrix is 1 if there is a lecture for that Teaching in that Slot, and 0 otherwise. The aim is to develop a solution that reduces the numerical complexity and run time with respect to the existing approaches. This goal requires a review of the hard and soft constraints.

The first phase of the thesis focuses on identifying the requirements of the problem. Some are inherited from the previous works, some are introduced, and some are not considered since they are not useful for finding a valid solution.

After the requirements analysis, the thesis provides a study of the technologies and the literature in order to find the models and approaches already available.

Following this study, the conclusion is that the best option is to use the CPLEX software, developed by IBM, which is capable of solving optimization problems. CPLEX is widely known for its performance and robustness, and the fact that it was used in the previous works represents an advantage. Initially, I generate timetables considering the courses related to the academic year 2023-24, in order to have a direct comparison between the timetables generated with this new approach and those generated in the previous theses. This requires refining the constraints to obtain compact timetables without too many consecutive lecture Slots or empty Slots.

Once satisfactory results are achieved, I retrieve the course data for the academic year 2025-26 and generate the timetables using those Teachings. The number of Teachings for the academic year 2025-26 is significantly larger than the number of Teachings for the academic year 2023-24, which requires another refinement of the constraints in order to generate timetables in a reasonable amount of time and with a good quality of the solutions.

The final results are encouraging. The generated timetable, obtained in a computational time of 13 hours, satisfies all hard constraints and is able to adequately handle soft constraints. The results therefore meet the needs of both Teachers and Students. Questionnaires are underway to identify any potential issues.

An important aspect to take into account during this work is code readability and cleanliness. Other thesis students, teachers, and collaborators have to be able to understand the code and modify it. For this reason, I use SonarQube to perform code reviews automatically and highlight potential issues in the code structure.

The final section of the thesis provides some suggestions for future improvements.

The work is completed by instruction manuals that describe how to use the tools developed.

Acknowledgements

Well, here we are. These five years have been challenging, made of joys and difficult moments. But none of the goals I have achieved would have been possible alone. I have to say thank you to all the people who have walked this path with me.

My parents, Loredana and Marco, who have given me moral and financial support throughout all these years.

My grandparents, Pietro, Maria, and Marisa, who accompanied me during this journey.

My friend Beatrice, who has always been by my side through the highs and the lows and always knew what to say to cheer me up when the burden of exams became too much.

My friends Andrea, Samuele, and Tommaso, who have given me a way to "escape" and find moments of lightness whenever I needed them.

My supervisors, Professors Renato Ferrero and Sophie Fosson, for their trust and guidance during this thesis.

And all my other friends and colleagues - too many to name all of them - who have faced this journey with me and who I hope will be by my side on many more.

To all of you, thank you.

Paolo

Ringraziamenti

Ebbene, eccoci qua. Questi 5 anni sono stati impegnativi, fatti di gioie e momenti difficili. Ma nessuno dei traguardi che ho raggiunto sarebbe stato possibile da solo. Devo ringraziare tutte le persone che hanno affrontato questo cammino insieme a me.

I miei genitori, Loredana e Marco, che mi hanno dato supporto morale ed economico durante questi anni.

I miei nonni, Pietro, Maria e Marisa, che mi hanno accompagnato durante questo viaggio.

La mia amica Beatrice, che è sempre stata al mio fianco attraverso gli alti e i bassi di questo percorso e sapeva sempre cosa dire per tirarmi su quando il peso degli esami diventava troppo.

I miei amici Andrea, Samuele e Tommaso, che mi hanno dato modo di "fuggire" e trovare momenti di leggerezza ogni volta che ne avevo bisogno.

I miei relatori, i professori Renato Ferrero e Sophie Fosson, per la loro fiducia e guida durante questa tesi.

E tutti i miei altri amici e compagni di università - troppi per nominarli tutti - che hanno affrontato questo percorso con me e che spero siano al mio fianco per molti altri.

A tutti voi, grazie.

Paolo

Table of Contents

List of Figures

Chapter 1

Introduction

The goal of this thesis is to develop a tool that supports the creation of the courses timetable for a university. In particular, as a case study, I consider the Computer, Cinema, and Mechatronic Engineering (ICM) and Electronic, Telecommunication, and Physical Engineering (ETF) Colleges of the Politecnico di Torino.

The timetable is subject to many constraints. Some constraints are related to the Students, e.g., mandatory Teachings cannot overlap, there cannot be too many consecutive hours of lectures or too many empty Slots between lectures. Other constraints are related to the Teachers, e.g. Teachers can specify Slots in which they cannot have any lecture, and to the single Teachings, e.g. the lectures of some Teachings should be allocated in two consecutive Slots at least once per Week. Among these constraints, I identify Hard Constraints, which must be respected, and Soft Constraints, which, if not respected, can introduce penalties. The goal is to find a timetable that respects all the Hard Constraints and maximizes an objective function built from the Soft Constraints.

Two theses were conducted on this topic, see [1] and [2], in which the timetable allocation was modeled as an ILP (Integer Linear Programming) problem and solved using the solver CPLEX. This thesis, also based on ILP, explores a novel methodology, which consists of representing the timetable as a two-dimensional matrix with the lecture Slots on the columns and the Teachings on the rows. Each element of the matrix is 1 if there is a lecture for that Teaching in that Slot, and 0 otherwise. The aim is to develop a solution that reduces the numerical complexity and run time with respect to the existing approaches.

The first phase of the thesis focuses on identifying the requirements of the problem. Some are inherited from the previous works, some are introduced, and some are not considered since they are not useful for finding a valid solution.

After the requirements analysis, the thesis provides a study of the technologies and the literature in order to find the models and approaches already available.

Initially, I generate a timetable considering only the hard constraints in order

to verify the validity of the model and that it can generate a solution within a reasonable amount of time. I work on the data related to the academic year 2023-24, since they are already available from the previous theses and allow a direct comparison between my results and those obtained in the previous works. In order to generate timetables faster, during the initial phase, I consider a subset of those data, which includes only the Mechatronic Engineering Degree Course. This allows the algorithm to generate a timetable in less than a minute[1].
I then proceed with a review of the timetable generated and I compare it with those of the previous theses to verify its validity and refine the hard constraints.

Afterwards, I introduce the soft constraints, which allow me to generate a more compact timetable, with fewer overlaps between Teachings and fewer empty Slots compared to the one with only hard constraints. As for the previous timetable, this one only considers the Mechatronic Engineering Degree Course, and the algorithm is able to generate it in around 10 minutes.
I review the new timetable and compare it with those generated in the previous works.

Once all the hard and soft constraints have been refined and the algorithm is able to generate a valid timetable for Mechatronic Engineering, I include all the Degree Courses of the ICT and ETF Colleges and generate a timetable considering all of them simultaneously. This expansion significantly increases the problem size, from 20 Teachings for Mechatronic Engineering to more than 320 for all the Degree Courses combined. As a result, both hard and soft constraints require another refinement in order to be able to find a feasible solution. Even after the adjustments, finding a timetable requires excessive time and the solutions found are not optimal, with many empty Slots or consecutive lecture Slots. For this reason, I explore a different approach, considering a subset of Degree Courses at a time and generating the timetable incrementally. With this strategy, the algorithm is able to find a solution in approximately 3 hours.

After comparing the results with those from the previous theses again and making the final adjustments to the constraints, I am ready to retrieve the data related to the academic year 2025-26. The data about those Degree Courses is stored in Excel files, so I use an algorithm to extract it and save it to a database, which is then used by the allocator algorithm.
One of the main challenges encountered with the data about the academic year 2025-26 is that the number of Teachings is significantly larger compared to the academic year 2023-24, which requires an additional review of constraints and parameters. After these refinements, the algorithm is able to generate a feasible solution in approximately 4 hours and 30 minutes.

[1]All execution times are based on a PC with a 16-threads processor and 32GB of RAM.

The last goal of this work is to include the Teachers' preferences about the Slots allocation for their lecture. The introduction of this constraint, modeled as a soft constraint, represents a challenge, since it increases significantly the number of soft constraints and therefore CPLEX tries to optimize the solution indefinitely without ever returning a result. In order to solve this problem, it is necessary to limit CPLEX execution time, and the final time needed to find a solution increases to 13 hours.

Finally, in order to know if the results can be applied in a real-world scenario, I prepare a questionnaire and send it to the Teachers and the Students' representatives, asking for their feedback on the generated timetable and any suggestions for further improvements.

Documentation has also been produced along with the code, with the aim of providing practical guidance both for those who wish to use the tool to generate a timetable and for those who wish to edit the code to add new constraints or modify the existing ones.

The thesis, in conclusion, provides some suggestions for future improvements. Among those, we have the exploration of a different methodology, not based on Integer Linear Programming or CPLEX, in order to generate the timetables (CPLEX has good performance with a limited amount of constraints and Teachings, but when those numbers increase it does not scale well), the avoidance of CPLEX running indefinitely when trying to optimize the solution, and a better Database organization.

Another important aspect to take into account during the development of the thesis is code readability and cleanliness. Other thesis students, teachers, and collaborators have to be able to understand the code and, if needed, modify it, so the code developed has to be as clean as possible and the technical debt has to be reduced to a minimum.
For this reason, I use the SonarQube tool to perform code reviews automatically and highlight potential issues in the code structure. These issues are divided into identity disharmonies (flaws that affect single entities), collaboration disharmonies (flaws that affect several entities at once), and classification disharmonies (flaws that affect entities tied by inheritance or abstraction).
By using SonarQube, I am able to maintain high code quality standards and effectively manage the technical debt.

1.1 Thesis structure

The rest of the thesis is organized as follows:

- Chapter 2 introduces the main concepts about Linear Programming and Constraint Programming, useful to understand the following chapters, as well

as the technologies and software used during the thesis.

- Chapter 3 describes the requirements for the timetable, as well as the hard and soft constraints implemented.
- Chapter 4 presents the design process behind the implementation of the algorithm and analyzes other software available on the market and the state-of-the-art.
- Chapter 5 provides an overview on how the algorithm has been implemented.
- Chapter 6 is a more practical chapter, that provides a guide on how to configure and use the algorithm.
- Chapter 7 describes the process to validate the generated timetable, and analyzes the performances of the tool.
- Chapter 8 provides further considerations about the results obtained, followed by possible future developments of the work.

Chapter 2

Background

This chapter introduces the main concepts to understand the thesis.

2.1 Operations research

Operations research is a field of applied mathematics which uses mathematical modeling, statistical analysis, and optimization techniques to solve complex problems and make informed decisions. The goal of using operations research is to make decisions that maximize outcomes and minimize costs.

Operations Research divides into three steps:

- Problem formulation, involving clearly defining the problem, identifying the variables, objective function, and constraints.
- Representing the problem in mathematical form, using linear or integer programming. Data collection and analysis represent a crucial part of this step.
- Optimizing the solutions found, in order to return the optimal or near-optimal one.

Operations research is used extensively in the economic, infrastructural, logistical, military, service, and transport fields. See [3] for more details.

2.2 Integer Linear Programming

According to [4], Linear programming (LP) is a method of achieving the optimal (or suboptimal) solution in a mathematical model in which all decision variables are continuous. Furthermore, in LP, the objective function and the constraints must consist of linear expressions. A particular case of LP is Integer Linear Programming

(ILP), in which all of the variables are restricted to be integers. During this thesis, I applied ILP to the timetable generation problem.

2.2.1 Linear expression

A linear expression is a scalar product; for example, the expression:

$$\sum a_i x_i$$

where a_i represents constants (that is, data) and x_i represents variables or unknowns. This expression can also be written in short form as a vector product:

$$^{t}AX$$

where A is the vector of constants and X is the vector of variables.

2.2.2 Linear constraint

A linear constraint is expressed by an equality or inequality:

- $linear\_expression = linear\_expression$
- $linear\_expression \geq linear\_expression$
- $linear\_expression \leq linear\_expression$

Note: strictly greater than or less than operators ($<$ and $>$) are not allowed in linear constraints.

2.2.3 Symbolic representation of an LP

Typically, a symbolic representation of an LP problem is:

$$min\ C^{\top}x$$

$$s.t.\ Ax \geq B$$

$$x \geq 0$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m,n}$, and $B \in \mathbb{R}^m$.

2.3 Numerical optimization solvers

As described in [5], optimization solvers solve mathematical programming models, constraint programming, and constraint-based scheduling models. They deal with generating and selecting the best solution among those that respect the given constraints. After an extensive analysis of the different optimization solvers available on the market, which is covered in detail in chapter 4, I concluded that the best option for this work is the CPLEX Optimizer.

2.3.1 CPLEX Optimizer

The CPLEX Optimizer, developed by IBM, is a high-performance numerical optimization solver. It is widely used for solving optimization problems, such as:

- LP problems.
- Mixed-Integer Programming problems.
- Quadratic Programming problems.
- Quadratically Constrained Programming problems.

The main characteristics that distinguish CPLEX among other numerical optimization solvers are its efficiency, reliability, and ability to handle large problems with millions of variables and constraints.

In this thesis, the CPLEX Optimizer software is used in combination with Python, via its Python language interface, DOcplex. Please refer to [6] for a detailed description of the software capabilities as well as tutorials on how to use it.

2.4 ILP model for timetable allocation

This section reports the main concepts related to the ILP model used for the Teaching timetable allocation.

2.4.1 Constraints

According to [7], in operations research, constraints refer to limitations, conditions, or restrictions that must be satisfied. In reality, they divide into hard constraints, which must always be respected, and soft constraints, which compose an objective function that should be maximized.

Key Characteristics:

- **Feasibility**: Constraints represents real-world limitations, to ensure that solutions to an optimization problem that respect these constraints are feasible in a real-case scenario.
- **Mathematical Representation**: Each constraint must have a mathematical representation in order to be implemented in the model. In LP they are expressed as equations or inequalities involving decision variables.
- **Role in Optimization**: Constraints help narrowing down the number of possible solutions and, therefore, the search space. By analyzing the constraints it is possible to exclude a portion of the search space, eliminate infeasible

solutions, and identify the most practical and viable solutions that meet all requirements.

- **Trade-offs**: Constraints often introduce trade-offs in optimization problems. You might not be able to respect all the constraints of the model at the same time, therefore it is important to understand the impact of each constraint on the problem and which have to be prioritized.

2.4.2 Objective function

In LP, an objective function is a linear function composed of the soft constraints that has to be minimized or maximized. An example of an objective function with two decision variables is:

$$Z = ax + by$$

in which x and y are the decision variables that represent the soft constraints and a and b are the penalties introduced if those constraints are not respected. The result, Z, has to be minimized (or maximized). We refer the reader to [8] for more details.

2.4.3 Types of problems and solutions

An *admissible solution* of an LP problem is an array $x \in \mathbb{R}^n$ that satisfies all the hard constraints.

The set of all admissible solutions is the *admissible region* or *admissible set*.

An *optimal solution* $x*$ is an admissible solution that minimizes or maximizes the objective function. A detailed description is available at [9].

In LP, we identify 3 types of problems:

- The problem is *inadmissible*: The set of admissible solutions is empty.
- The problem is *unlimited*: It is possible to find admissible solution that minimize (or maximize) the objective function without any limit.
- The problem *allows an optimal solution*: There is at least one admissible solution that optimizes the objective function.

2.5 Technical debt

In software development, technical debt is a metaphor introduced by Ward Cunningham and refers to the costs of additional work and potential problems that arise from choosing an expedient solution in the short term but suboptimal or unsustainable in the medium-long term.

Similarly to financial debt, technical debt can accumulate "interest" over time and lead to increased maintenance costs, reduced code quality, and slower development velocity. Technical debt is not always negative; in fact, it can be a useful resource that allows companies to deliver a product over a short period of time. However, it needs to be carefully managed and repaid in a reasonable amount of time, in order to maintain software quality and long-term sustainability. Managing technical debt is an activity that involves identifying, tracking, and prioritizing areas that require improvement, and a portion of the development time should be assigned to technical debt repayment. Please refer to [10] for more details.

2.5.1 SonarQube

As described in [10] and [11], SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality. It analyzes and measures source code in terms of code reliability, security vulnerabilities, and maintainability issues (or code smells). SonarQube produces reports each time the codebase is updated, classifying the issues in the code into high, medium, and low priority. With these reports, developers can easily track the technical debt in their code and effectively manage it.

2.6 Terminology

The final section of this chapter introduces the terminology related to the Degree Courses and the timetable allocation of the Politecnico di Torino, which will be used throughout the rest of this thesis.

- **College**: College refers to an internal organization responsible for the coordination and management of the educational activities related to a specific cultural and disciplinary area. As a case study for the timetable allocation, I am interested in the Computer, Cinema, and Mechatronic Engineering (ICM) and Electronic, Telecommunication, and Physical Engineering (ETF) Colleges.
- **Degree Course**: A Degree Course is an academic program offered by the university that leads to the award of a specific academic degree (Bachelor's or Master's). Each degree is composed by one or more Orientations.
- **Orientation**: Within a Degree Course, an Orientation is a disciplinary path that Students can choose in order to specialize in a specific area of their Degree Course.
- **Teaching**: A Teaching is a single educational unit. It corresponds to a subject or topic taught during one or more semesters and is associated with one or

more Teachers, a certain number of Slots, and one or more Orientations. All Teachings have at least one Theory lecture Slot, and can have Practice lectures and Laboratory lectures.

- **Teacher**: A Teacher is a member of the academic staff responsible for delivering one or more Teachings within a Degree Course. Teachers at the Politecnico di Torino may hold different academic roles, such as professor, associate professor, or researcher.
- **Slot**: A Slot is a single, indivisible lecture unit, that lasts 1 hour and 30 minutes. Each lecture of each Teaching is allocated in one or more Slots. There are 7 Slots per Day (8:30-10:00, 10:00-11:30, ..., 17:30-19:00).
- **Day**: Day refers to the days in which students can have lectures. There can be 5 (Monday-Friday) or 6 (Monday-Saturday) days of lectures per Week. As mentioned, there are 7 Slots per Day, except for Saturday, which can have 1-7 lecture Slots.

Chapter 3

Problem definition

The timetable allocation problem is modeled as an ILP problem. The timetable has to respect a list of constraints, divided into hard and soft constraints. With respect to [1] and [2], I update this list. In particular, I add some significant constraints, while I remove the less useful ones to reduce the numerical complexity. Here I provide a detailed analysis of the timetable requirements and the constraints used, as well as those that have been removed or modified. The reason behind the removal or the edit of some of the constraints used in the previous theses is to be found in the desire to obtain a lighter and more efficient model.

3.1 Requirements

Requirements represent the high-level considerations that have been made before generating the timetable. They are necessary in order to design the model.

3.1.1 Inherited requirements

I report here the requirements inherited from the previous theses:

- "There are Colleges[1], Degree Courses, Orientations, Teachings, Slots, Days, and Teachers".
- "The Degree Courses are divided in Master's Degree and Bachelor's Degree".
- "A Degree Course has different Orientations[2]".

[1] ICM and ETF

[2] There is more than one orientation for each Degree Course

- "An Orientation has different Teachings".
- "A Teaching can be in different Orientations".
- "An Orientation is composed by different years, 3 for Bachelor's Degree Orientations and 2 for Master's Degree Orientations".
- "The Teachings in an Orientation have a certain Correlation between them, which can vary between 0 and 100[3]".
- "There can be different paths in an Orientation[4]".
- "Each Teaching is associated to one or more Semesters and to one Year for each Orientation".
- "A Teaching has one or more Slots".
- "A Teaching is associated to one or more Teachers and one of them is the Main Teacher".
- "A Teaching has a fixed allocation plan for the whole semester".
- "A Teaching is part of a College".
- "A Slot can be a Theory lecture, Practice lecture, or Laboratory lecture".
- "A Teaching can have Slots in which the students are divided in groups".
- "One or more Teachers are associated with each Slot".
- "Each Slot is in a Day of the Week: Monday to Friday and eventually on Saturday".
- "Each Slot consists of 1.5 hours and is in a Time Slot, between 8:30 and 19:00".
- "Each Teacher can specify up to 4 Slots in which they are unavailable".
- "Teachers might have more unavailabilities related to Teachings in other Colleges".
- "For each Teaching a weekly allocation plan is defined".

[3]The Teachings in a Year do not have the same importance. There are Teachings that are mandatory and others that can be chosen. The correlation is a number that express how two Teachings are related to each other

[4]A path is the set of Teachings chosen by a student. It is important to note that not all students choose the same Teachings.

- "Students choose and Orientation and a study plan[5]".
- "Two Teachings can:
  - Never overlap: hard constraint.
  - Partially overlap: soft constraint.
  - Overlap completely: no constraints."
- "An Allocation Plan corresponds to the weekly timetable of all the Teachings, containing all their Slots with the information such as Day, Time Slot, Teachers, Type, and Group".
- "There is the Allocation Plan related to the previous year[6]".

3.1.2 New Requirements

Here I report the requirements that were not considered in [1] and [2] and are added during this work:

- "The Main Teacher of a Teaching can express preferences regarding the allocation of Slots for that Teaching[7] (for example, there should be at least one double Slot[8] during the Week, there should be no double Slots during the Week, there should be only double Slots)".
- "The number of consecutive lecture Slots should be limited".
- "The number of empty Slots between two lectures should be limited".
- "The number of Students who have lecture at 8:30 and at 17:30 in the same Day should be reduced to the minimum".

These new requirements take into account the Students' needs about consecutive lecture Slots and empty Slots, as well as the Teachers' needs about their courses' Slots organization.

[5]By choosing non mandatory Teachings you can generate a study plan, which is an instance of a path

[6]This requirement was considered in the first thesis but not in the second one

[7]In other words, the Main Teacher can express a preference on how the Theory, Practice, and Laboratory lecture Slots of its Teachings should be organized.

[8]A double Slot are 2 consecutive Slots of the same lecture.

3.1.3 Other Requirements

Here there is a list of requirements that were used in [1] and [2] but not in this work, since they are not considered useful for finding a valid solution.

- "For each Teaching a Timetable Template is defined".
- "A Teaching is attended by a certain number of students".
- "Rooms can be Classrooms or Laboratories. Classrooms are divided in *Aule attrezzate CA2*, *Aule attrezzate CA*, *Aule TableBox*, *Aule WallBox*, and *Aula 5T*. Laboratories are: *LABINF*, *LAIB*, *LADISPE*, *LED*, *LED1*, *LED2*, and *ACSLAB*".
- "Slots must be allocated in a specific Room type".
- "Classrooms have a maximum capacity. This is in relation with the number of students that attend a Teaching".
- "Each Slot is related to a Room".
- "Teachers can express preferences about the Slots in which they want to have lectures".

The requirements that are not considered are those related to the Rooms allocation and the Teachers' preferences about the Slot in which they would like to have lectures. The decision to exclude these constraints is due to the desire of reducing the numerical complexity of the model.

3.2 Correlations

During the previous theses, in order to model the relation between two Teachings and define whether or not they could overlap, the Teachings were divided into categories (*Mandatory*, *Mandatory - chosen from table*, *Suggested*, *Choice from table*, *Free Credits*) and the concept of Correlation was introduced.

During this work, the Teachings categories are reduced, in order to have a lighter model. Those that are maintained are:

- **Mandatory**: A Teaching that must be followed by all Students enrolled in a specific Orientation.
- **Mandatory, chosen from table**: This is a type of semi-compulsory course. When filling their Study Plan, Students are required to select one or more Teachings from a predefined list (for example, Students can choose a Teaching in Italian or in English).

- **Free Credits**: Students can freely allocate these Teachings so that they can personalize their study plan according to their interests.

The Correlation is a value that can vary between 0 and 100 and expresses how two Teachings compare to each other. A Correlation of 100 means that those Teachings are chosen by the vast majority of Students and should never overlap (hard constraint). A Correlation between 1 and 99 means that the two Teachings can overlap, but if they do, a penalty proportional to the Correlation value is introduced (soft constraint). Lastly, a Correlation of 0 (or no Correlation) means that little-to-no students choose both of the Teachings in the same semester and therefore can completely overlap (no constraint).

In Table 3.1 I report the correlation values that I used to build the model.

	Mandatory	Mandatory, chosen from table	Free credit
Mandatory	100	100/n_teachings_in_table	20
Mandatory, chosen from table	100/n_teachings_in_table	100/n_teachings_in_table	20
Free credit	20	20	20

Table 3.1: Correlations between Teachings

3.3 List of Constraints

In order to be able to satisfy the requirements, they have to be implemented in the allocator in the form of Constraints. Since this thesis uses a different approach than those explored in the previous two, I decided to rewrite the Constraints instead of starting from those implemented in the previous works.
I divide the Constraints between hard and soft constraints, and they are reported here:

3.3.1 Hard Constraints

Teachings:

- **Slots per Week**: Each Teaching must have the exact amount of Theory, Practice, and Lab Slots per Week specified in the Teaching's description.
- **Number of consecutive Slots**: Each Teaching must have at most 2 Slots in a Day and, if so, the two Slots must be consecutive.

- **Limited number of Teachings in a Day**: The number of Correlated Teachings in a Day is limited, in order to not have too many consecutive lectures in the same Day.
- **Overlaps**:
  - Teachings with a Correlation above a certain threshold must never overlap. Furthermore, Mandatory Teachings must never overlap with other Teachings, regardless of their type.
  - Different Groups of Practice lectures of the same Teaching cannot overlap, since there might be only one Teacher for all the Practice Groups.
  - Same as above but for Laboratories.
  - The same groups of Practice and Laboratory lectures of the same Teaching cannot overlap (e.g. Practice Group1 of TeachingA cannot overlap with Lab Group1 of TeachingA, but Practice Group1 of TeachingA can overlap with Lab Group2 of TeachingA only if the Practice and Laboratory Teachers are different).
  - The same Groups of Practice lectures of different Teachings with a Correlation above a certain threshold cannot overlap (e.g. Practice Group1 of TeachingA cannot overlap with Practice Group1 of TeachingB, but Practice Group1 of TeachingA can overlap with Practice Group2 of TeachingB).
  - The same Groups of Laboratory lectures of different Teachings with a Correlation above a certain threshold cannot overlap (e.g. Lab Group1 of TeachingA cannot overlap with Lab Group1 of TeachingB, but Lab Group1 of TeachingA can overlap with Lab Group2 of TeachingB).
- **Correlation between first and last Slot of the Day**: The sum of the Correlation of the Teachings in the first and last Slot of the Day must be under a threshold, in order to minimize the number of Students who have lectures at 8:30 and at 17:30 on the same Day.

Teachers:

- **Overlaps**: Teachings taught by the same Teacher must not overlap (considering Theory lectures, Practices, and Labs).
- **Unavailable Slots**: Each Teacher can indicate up to 4 Slots in which they are unavailable. Their Teachings cannot be allocated in those Slots.

3.3.2 Soft Constraints

- **Difference between the first and last lecture of a Day**: The difference between the first lecture Slot of a Day and the last one should be minimized (while maintaining some empty Slots during the Day), in order to have a more compact timetable and avoid too many empty Slots.
- **Overlaps**: The overlaps between Teachings with a Correlation below the threshold should be minimized.
- **Teachers' preferences**: As said previously, a Teaching's Main Teacher can express a preference about the Slots allocation. The amount of Slots (whether they are Theory, Practice, or Laboratory Slots) that respect these preferences should be maximized.
- **Distance between different Groups of the same Practice/Laboratory**: Where possible, Practice or Laboratory Slots of different Groups for the same Teaching in the same Day should be consecutive. For example, considering the Laboratory Groups for Teaching A, it would be ideal to have Group 1 in Slot 1 and Group 2 in Slot 2, so that students from Group 1 can stay for the Group 2's lecture if needed.

Chapter 4

Design

This chapter analyzes how the timetable generation problem has been represented and how I modeled the constraints.

4.1 Data sources

One of the main problems in the previous theses was the heterogeneity of the data sources. The previous allocator worked with Excel files, CSV files, and a Database. So, in order to have cleaner code, I opted to merge all those data into one single Database called Courses_DB, which is used throughout the whole project.

Using the script *Excel_to_db_converter*, I insert all the information from the various Excel files into the database.

4.1.1 Database Courses_DB

This database contains all the data needed by the allocator to generate the timetable, as well as the results of the computation. The Italian names, as well as the inconsistencies in the naming system, are due to the fact that many of the tables in the database had already been defined in the previous theses, and I added new tables during this work.

Here I report a list of the tables, in alphabetical order:

- **Corso_di_laurea**: This table contains the names of the Degree Courses for which I generate the timetable.
- **Docente**: The table contains the information about the names and IDs of the Teachers of all the Degree Courses taken into account.
- **Docente_in_Insegnamento**: This table associates each Teacher with their Teaching(s), specifying the number of hours (nOre) they have on that Teaching

and the lecture type (tipoLez; it can be L - Theory lecture, EA - Practice lecture, or EL - Laboratory lecture).

- **Docente_in_Slot**: Once the timetable is generated, the lecture Slots for each Teacher are saved in this table.

- **Info_correlazioni**: Here there are the correlations between Teachings. The column Correlazione (Correlation) represents the Correlation extracted from the data sources.
  In the previous thesis, the possibility to correct the Correlations by hand was introduced. In this case, they are saved in the column Correlazione_finale (Final Correlation) (I do not use this option during this thesis).
  A third column, Obbligatorio (Mandatory), is useful to know if one of the two Teachings in the Correlation is mandatory and, therefore, cannot overlap with the other one.

- **Insegnamento**: This table contains the information about each Teaching, including the Main Teacher, the hours of Theory, Practice, and Laboratory lectures, the Teachers' preferences about the allocation of those hours (n_min_double_slots, n_min_single_slots, n_min_double_slots_practice, n_min_single_slots_practice), and the information about the allocation of the Laboratory's blocks (double_slots_lab = 1 if the Laboratory blocks should consist of double Slots, 0 otherwise).

- **Insegnamento_in_Orientamento**: This table associates the Teachings with the Orientation(s) they belong to.

- **Insegnamento_listCodIns**: This table associates the IDs of the Teachings with their ID_INC (ID Incarico, different from the Teachings' IDs because the same Teaching can have different IDs in different Orientations, but only one ID_INC).

- **Orientamento**: This table contains the information about the Orientations of each Degree Course.

- **PianoAllocazione**: The list of the generated timetables is stored here. You can also provide a description for each of them.

- **PreviousSolution**: This table contains a previous timetable that can be used by CPLEX as a base to start from when generating a new one.

- **Slot**: This table contains the information about the Slots of a generated timetable. For each Teaching, I save its Slots here.

- **SlotSettimana**: The table saves the information about the Slots in a Week (for example, Mon. 8:30-10:00). This is used by the GUI Web Application when representing the timetable.
- **Teachers_Unavailability**: This table contains the Slots in which a Teacher is not available.

4.1.2 Excel files

File PreferenzeDocenti.xlsx

This file is extracted from the university's portal and contains the preferences of the Main Teachers about their Teachings' Slot allocation, as well as the Teachers' unavailabilities. Note that the column names are in Italian. I am interested in these columns in particular:

- **NUM_ORE_TOT**: The total number of hours of a Teaching in a semester, considering lectures, Practices, and Laboratories.
- **NUM_ORE_ESE**: The total number of Practice hours in a semester.
- **NUM_SQU_ESE**: Number of Practice groups.
- **NUM_ORE_LAB**: The total number of Laboratory hours in a semester.
- **NUM_SQU_LAB**: Number of Laboratory groups.
- **ORGANIZZAZIONE_BLOCCHI_LEZIONE**: Contains the Teacher's preference about the allocation of the lecture Slots. Its values can be: "tutti i blocchi da 3h" (all blocks of 3h), "un blocco da 3h e gli altri da 1,5h" (one block of 3h and the others of 1.5h), "un blocco da 4,5h (Atelier per Architettura)" (one block of 4.5h, Atelier for Architecture).
- **ORGANIZZAZIONE_BLOCCHI_ESERCITAZIONE**: Contains the Teacher's preference about the allocation of the Practice Slots. Its values can be: "tutti i blocchi da 1,5h per ciascuna squadra" (all blocks of 1.5h for each group), "tutti i blocchi da 3h per ciascuna squadra" (all blocks of 3h for each group), "un blocco da 3h e gli altri da 1,5h per ciascuna squadra" (one block of 3 hours and the other of 1.5h for each group). NOTE: in some cases the Teachers request blocks of 3 hours (2 Slots), but there is only one Slot per Week. In that case the algorithm allocates blocks of 1.5 hours (1 Slot).
- **NUM_BLOCCHI_SETTIMANALI_LAIB_ATENEO**: Number of blocks per Week for the Laboratory.

NOTE: this can be empty, but the information about the blocks per Week could be found in the column NUM_BLOCCHI_SETTIMANALI_LAB_DIPARTIMENTALE.

- **NUM_SQUADRE_SETTIMANALI_LAIB_ATENEO**: Number of Laboratory groups.

  NOTE: this can be empty, but the information about the Laboratory groups could be found in the column NUM_SQUADRE_SETTIMANALI_LAB_DIPARTIMENTALE.

- **ORGANIZZAZIONE_BLOCCHI_LAIB_ATENEO:** Preference about the Slot allocation for the Laboratories. Its values can be: "blocchi da 1,5h per ciascuna squadra" (blocks of 1.5h for each group), "blocchi da 3h per ciascuna squadra" (blocks of 3h for each group), "indifferente" (no preference).

  NOTE: this can be empty, but the information about the organization of the Lab Slots could be found in the column ORGANIZZAZIONE_BLOCCHI_LAB_DIPARTIMENTALE.

- **INDISPONIBILITA_SETTIMANALI**: Slots in which the Teacher is unavailable (max. 4 per Teacher).

NOTE: some Teachers do not express their preferences for their Teachings here. Therefore, in order to retrieve the number of Practice and Laboratory hours, I have to use the columns h_ese and h_lab of the Degree Courses files (see 4.1.2 for more details).

File "Percorsi-gruppi-insegnamenti aa 2026.xlsx"

In this file, which is also extracted from the university's portal, there is all the information about the Teachings for the academic year 2025/26. This file does not only contain the Degree Courses associated with ICM and ETF Colleges, but also those from other Colleges. So, before extracting the data, the script has to select only the Courses related to ICM and ETF. As for the previous file, column names are in Italian. The columns in which I am interested are:

- **ID_COLLEGIO**: Contains the ID of the College (ICM or ETF).
- **TIPO_LAUREA**: Degree type. Can be 1 (Bachelor's Degree) or Z (Master's Degree).
- **NOME_CDL**: Name of the Degree Course.
- **DESC_ORI**: Name of the Orientations in a Degree Course.

- **PERIODO_INI**: Didactic period in which the Teaching starts (can be 1 or 2). If the Teaching can be chosen from a table, the didactic period can be found in the columns PERIODO_INI_S or PERIODO_INI_SS.
- **ANNO**: Year of the Teaching (can be 1, 2, or 3).
- **COD_INS**: Teaching ID. The same Teaching can have different IDs in different Orientations.
- **TITOLO**: Name of the Teaching. There are some Teachings (Challenge, Thesis, Internship, etc.) that the script should ignore. If the Teaching can be chosen from a table by the students, this column will contain the name of the table (for example, "Insegnamento a scelta da tabella A" and the Teaching's name can be found in the columns TITOLO_S or TITOLO_SS).
- **CFU**: Number of credits of the Teaching.
- **ID_INC**: The unique identifier of the Teaching (unlike the Teaching ID, it does not depend on the Orientation).
- **MATRICOLA**: The ID of the Main Teacher of the Teaching.

Folder Courses Data

The Courses Data subfolder contains information about the Teachings. Here, there is one Excel file for each Degree Course. The main columns in these files are:

- **h_lez**: Number of hours of Theory lectures of a Teaching, not to be confused with the field "NUM_ORE_TOT" of the file "PreferenzeDocenti.xlsx", which includes Theory, Practice, and Laboratory hours.
- **h_ese**: Number of Practice hours and groups, in format: TYPE h_practice*n_groups*n_teache
  NOTE: h_practice refers to the Practice hours for each group individually.
- **h_lab**: Number of Laboratory hours and groups, in format: TYPE h_lab*n_groups*n_teachers
  NOTE: h_lab refers to the Laboratory hours for each group individually.
- **id_inc**: The unique identifier of the Teaching.
- **matricola**: ID of the Main Teacher of the Teaching.
- **Collaboratori**: Contains the information about the collaborators Teachers for the Teaching (i.e. the Teachers other than the Main Teacher), the type of their lectures, and their hours.

4.2 Lecture scheduling software: the state of the art

This section provides an analysis of the literature in order to understand if there are other solutions to the lecture allocation problem and how they have been implemented.

4.2.1 Genetic Algorithm

A widely used approach when generating lecture timetables exploits Genetic Algorithm. This type of algorithm, based on evolutionary biology techniques such as heredity, mutation biology, and natural selection, can be applied to solve optimization problems that are not well suited for standard algorithms, especially those where the objective function is not linear. These algorithms are often more efficient when compared to traditional ones, with a higher success rate in finding the best optimal solution. See [12] and [13] for more details.

I report here a high-level description of how a genetic algorithm works:

Produce the initial population of individuals

Evaluate the fitness of every individual

While termination condition not satisfied do

{

Select individual for reproduction

Recombine between individuals

Mutate individuals

Evaluate new solutions for fitness of modified individuals

Generate a new population

End while

}

Figure 4.1: Genetic Algorithm description

The flow diagram of the algorithm in Figure 4.1 is:

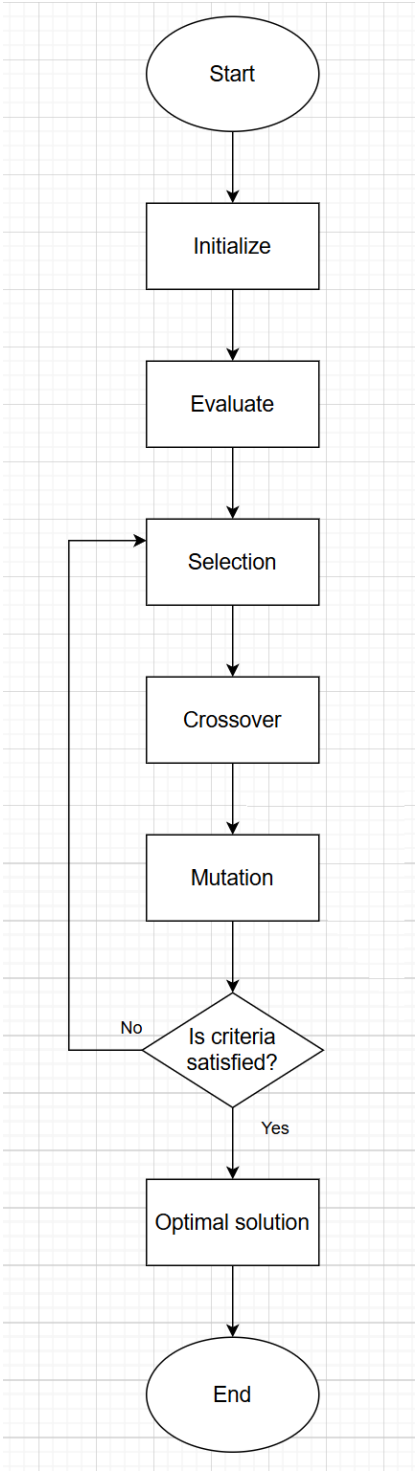

Figure 4.2: Genetic Algorithm diagram

4.2.2 Ant Colony System Algorithm

An alternative to Genetic Algorithm, when solving the lecture allocation problem, is the Ant Colony System Algorithm. This algorithm is used to solve optimization problems such as the traveling salesman problem. It is part of the metaheuristic algorithms, a family of algorithms that imitate social behavior or strategies that exist in nature.
This algorithm chooses a path in a similar way as ants move to search for food. Ants initiate the search by moving randomly around their nests. This opens up multiple paths from the nest to the food. Once the food is found, a portion of it is carried back to the nest, leaving a trace of pheromones on the return path, whose concentration depends on food quantity and quality. The probability of other ants following the path depends on the pheromone concentration and its evaporation. Please refer to [14] for more details.

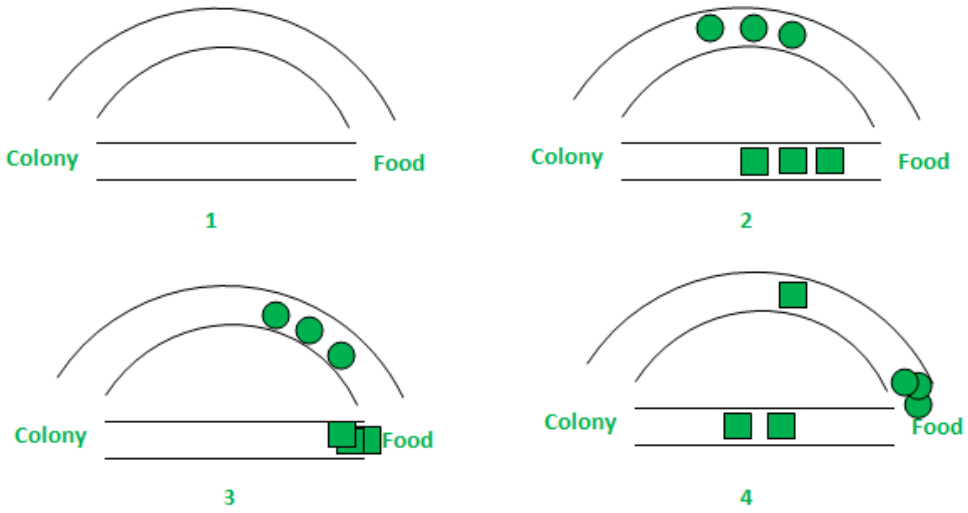

Figure 4.3: Ant path search

In the figure above, for simplicity, I consider only one food source and two possible paths. Initially, the ants split equally between the two paths in order to reach the food. When they have to return to the colony, in order to choose the path to take, they base their decision on the concentration of pheromone. Since the straight path is shorter, the pheromone concentration will be higher (less pheromone will have evaporated), leading more ants to choose that path and increasing the pheromone concentration.

The Ant Colony Algorithm simulates this behavior via weighted graphs where the ant colony and the food source are vertices, the paths are the edges, and the

pheromone traces are the weights of the edges.

Let $G = (V, E)$ be the graph, where V and E are the edges and vertices of the graph. We consider two vertices, V_s the vertex representing the ant colony (source) and V_d the vertex representing the food source (destination). We have two edges, E_1 and E_2, with lengths L_1 and L_2 and pheromone values R_1 and R_2. Therefore, the starting probability of an ant selecting a path between E_1 and E_2 can be expressed as

$$P_i = \frac{R_i}{R_1 + R_2};\ i = 1, 2$$

Now, while returning through the shortest path E_i, the pheromone value is updated for that path, based on the length of the path as well as the evaporation rate.

Update of R_i according to the length of the path (K is a parameter of the model):

$$R_i \leftarrow R_i + \frac{K}{L_i};\ i = 1, 2$$

Update of R_i according to the evaporation rate $v \in (0, 1]$:

$$R_i \leftarrow (1 - v) * R_i;\ i = 1, 2$$

At each iteration, the ants are in the source vertex V_s and move towards the destination V_d. After reaching V_d, they return to the source, choosing the return trip. A detailed description is available at [15].

The high-level description of the algorithm is:

Initialize necessary parameters and pheromone trials

While termination condition not satisfied do

{

Generate ant population

Calculate fitness values associated with each ant

Find best solution through selection methods

Update pheromone trial

End while

}

Figure 4.4: Ant Colony Algorithm description

4.2.3 FET Scheduling Software

There are many softwares on the market that are based on the two types of algorithms presented above. The majority of them require a yearly subscription in order to be used, and prices vary from 8.000€ to 12.000€, but there are some free alternatives as well.

Among the free alternatives, the one that I consider the most complete is the FET Scheduling Software. FET is a software written in C++ that uses the Ant Colony System Algorithm to generate timetables. It has very good performance: it can solve simple timetables in under 5 minutes, complicated timetables in 5-20 minutes, and extremely difficult timetables in hours. More details can be found at [16].

When using FET, you can specify the Teachings, the Teachers, the Classrooms, and the constraints between these three entities, and the software calculates a timetable based on those parameters.

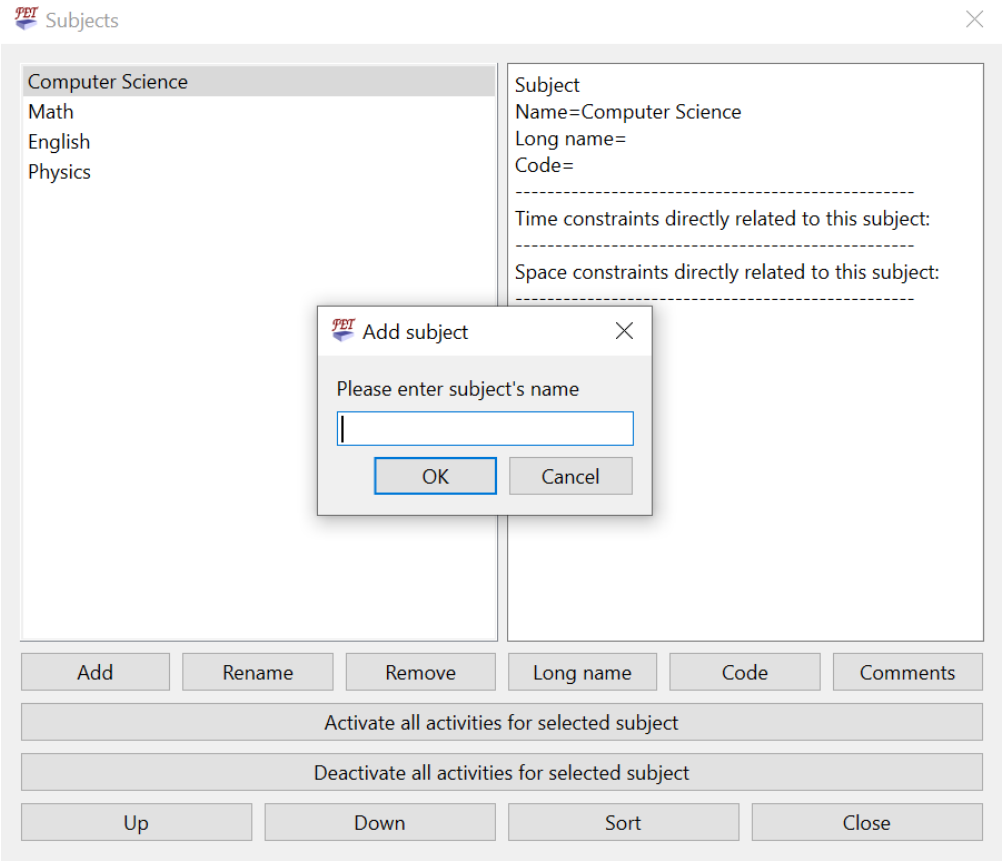

Figure 4.5: Adding a subject in FET

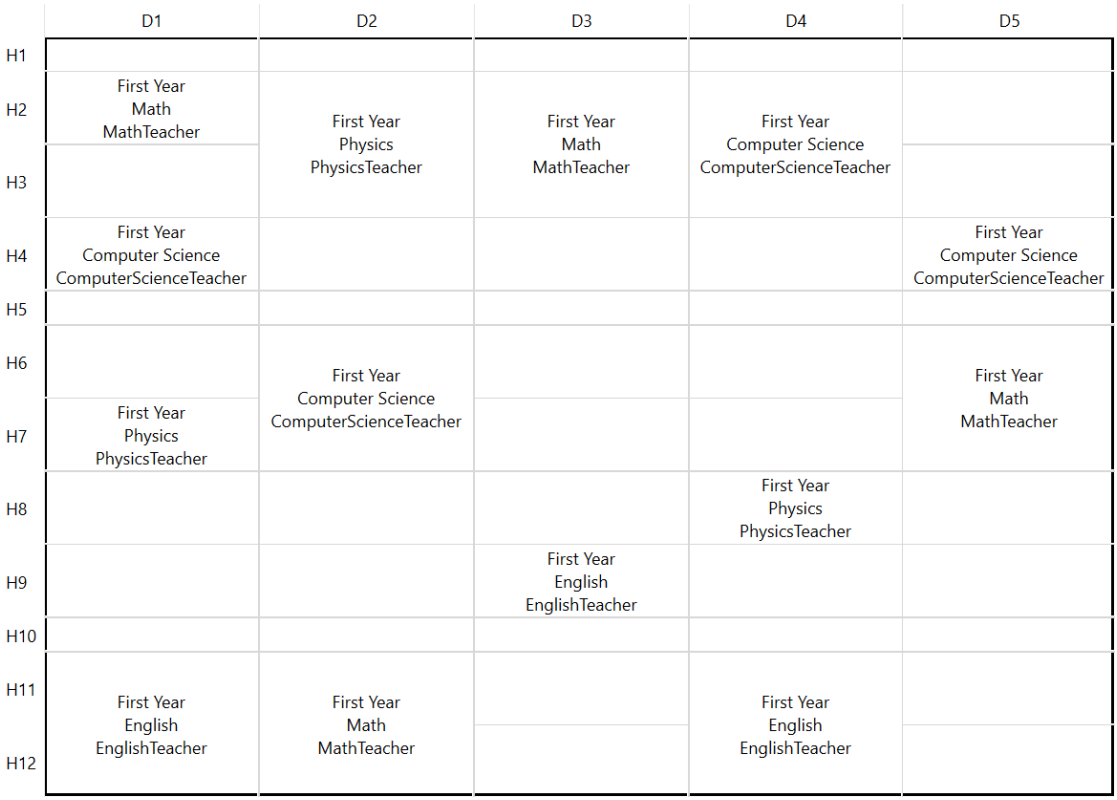

Figure 4.6: Timetable view in FET

4.2.4 Algorithm Choice

After an extensive analysis of the techniques mentioned above and the software available on the market, the conclusion is that, while the performances can be better when compared to the ILP methodology, one big problem common to the Genetic Algorithm and the Ant Colony Algorithm is that it is difficult to specify complex constraints such as the Teachers' preferences about lecture organization and the personalization options are limited.

Therefore, I believe that building a custom, ad-hoc algorithm using ILP techniques is the best option. I also made this decision because ILP was used in the previous theses, so there is already a basis to start from.

4.3 Optimization Softwares Analysis

After deciding the technique to use, I evaluate different optimization software. The previous theses both use CPLEX, but I want to explore other possibilities as well. In this section, I analyze the software evaluated and their pros and cons.

4.3.1 CBC

CBC (COIN-OR Branch and Cut), developed by the COIN-OR foundation, is an open-source solver for LP and Mixed Integer Programming (MIP) problems, written in C++.

Pros:

- Free and open source.
- Modifiable, the full source code is available.
- Easy to learn and use.
- Lightweight, has a small footprint on memory.

Cons:

- Limited scalability: While it works well with small problems, it becomes too slow when the problems' complexity grows.
- No parallelism: CBC does not support multi-threading natively.
- Does not support Constraint Programming

One of the main strengths of CBC is the fact that it is open source. However, the limited scalability and the limitations in further development and support represent a big obstacle. See [17] for more details.

4.3.2 GLPK

GLPK (GNU Linear Programming Kit) is an open-source solver for LP and MIP problems. It is part of the GNU project.

Pros:

- Free and open source.
- Easy to learn and use.
- Lightweight, has a small footprint on memory.

Cons:

- Works well with small and medium problems, but does not scale well with large models.

- No parallelism: GLPK does not support multi-threading natively.
- The project is relatively old, the last update was in 2012 and the community support is limited.

Similarly to CBC, GLPK is open source as well. The performance on medium-size models is marginally better than that offered by CBC, but performance on large models still represents a problem. A detailed comparison between CBC and GLPK is available at [17].

4.3.3 Gurobi

Gurobi, developed by Gurobi Optimization, LLC, is a proprietary, high-performance solver used for LP, MIP, and Constraint Programming problems.

Pros

- Very high performance, Gurobi is one of the best-in-class solvers and scales well with complex problems.
- Fast updates, Gurobi frequently releases new features and updates.
- Multi-threading, the algorithm supports multi-core CPUs natively.
- Gurobi provides extensive documentation, and community support can be easily found over the Internet.
- Academic license. Although Gurobi is a proprietary software, it is available free of charge for academic use.

Cons

- For the problem definition, Gurobi relies entirely on external APIs, it has no built in modeling language.
- More complex to use than other softwares.
- Gurobi's licenses are expensive for non-academic or non-research uses.

Out of the software analyzed up to this point, Gurobi is the most suitable one for complex problems with many constraints and variables, such as the timetable allocation. Its scalability and multi-threading support drastically improve the performance when compared to other solvers. Please refer to [18] for an extended analysis.

4.3.4 CPLEX

CPLEX, used in the previous two theses, is a proprietary software developed by IBM. It supports LP, MIP, Constraint Programming, and more.

Pros

- Top-tier performance and robustness. Just like Gurobi, CPLEX offers very high performances and works well even with complex problems.
- Includes algorithms such as parallel branch & bound, presolving, and heuristics, that reduce the research space and, therefore, the time needed to generate a solution.
- Similarly to Gurobi, CPLEX supports multi-threading natively.
- CPLEX offers many customization options. When executing the softwares, there are many parameters that can be changed in order to modify CPLEX's behaviour and how it utilizes RAM and CPU.
- CPLEX is widely used across the world, therefore many support and guides can be found online. Above that, you can find comprehensive documentation on IBM's website.
- Like Gurobi, CPLEX provides academic license.
- This algorithm was used during the previous thesis, so using it would provide a direct comparison on performances and results with the models implemented in the previous works.

Cons

- CPLEX is Closed-Source, which means the source code cannot be seen or modified.
- This Software has a larger installation footprint when compared to others.

CPLEX is widely known for its performance and robustness, which are very similar to those provided by Gurobi. Furthermore, the fact that it was used during the previous works represents a big advantage over other alternatives. Refer to [18] for a comparison between CPLEX and Gurobi.

For this reason, after evaluating the pros and cons of each solver, I consider CPLEX to be the best one for this thesis. Therefore, I am using this software via its Python API, DOcplex.

After choosing the solver, I report here the evaluation of two Python models that can help to define optimization problems in an easier way, and then export them and solve them using CPLEX. More details are available at [19].

4.3.5 PuLP

PuLP is a modeling library for LP and MIP. It allows users to define and implement optimization problems, which can then be solved using external software, such as CPLEX or GLPK.

Pros:

- By using this modeler, the problem definition is easier when compared to modeling the problem directly in CPLEX.

Cons

- For complex problems, CPLEX's API is more complete and has better performance.
- PuLP offers less functionalities when compared to CPLEX's API.

4.3.6 Google OR-Tools

OR-Tools is an open-source software suite for solving optimization problems developed by Google. As PuLP, it allows you to model the problem in the programming language of your choice and use an external solver to solve it.

Pros

- More powerful than PuLP, allows users to model more complex problems.
- It is actively maintained, there are frequent updates and the documentation is complete.

Cons

- Even though it is more powerful than PuLP, it still does not allow the representation of a problem as complex as the timetable allocation.

After evaluating those tools, the conclusion is that they are not powerful enough to model the timetable allocation problem. Even though they would make the modeling easier, their limitations can be an obstacle with more complex constraints. Therefore, I exclude them and decide to model the problem by using CPLEX's API for Python.

4.4 Adopted model

Along with the choice of the solver, another important phase of this thesis is the definition of the model used to represent the problem. During the previous theses, two models were explored: the Multidimensional Matrix Model and the Slots Model. Refer to [1] and [2] for a detailed description. This work explores a third model: the Bidimensional Matrix Model.

4.4.1 Bidimensional Matrix Model

As the name suggests, this model is based on a bidimensional matrix. On the columns of this matrix, there are the weekly Slots (Monday 8:30-10:00, Monday 10:00-11:30, etc.) and on the rows, there are the Teachings. Each cell of the matrix is 1 if the Teaching has a lecture in that Slot, and 0 if there is no lecture. This model simplifies the generation of the timetable, since there are fewer factors to take into account when compared to the Slots Model and the Multidimensional Matrix Model, but may not be suitable to implement a more complex allocator that also takes classrooms into account.

Variables

- **Teachings** index $t \in [0, nTeachings) = T$, T is the set of all the Teachings[1].
- **Teachers** index $i \in [0, nTeachers) = I$, I is the set of all the Teachers[2].
- **Slots** index $s \in [0, nSlot) = S$, S is the set of the weekly Slots. nSlot can vary between 35 and 42, according to the number of Slots that the user wants to allocate on Saturday.
- **Days** index $d \in [0, nDays) = D$, D is the set of Days in a Week. nDays can vary between 5 and 6, depending on whether the allocation on Saturday is enabled or not.
- **Timetable Matrix**: boolean variables $timetable\_matrix_{t,s} \in \{0, 1\}$:

$$timetable\_matrix_{t,s} = \begin{cases} 1, \text{lecture of Teaching } t \text{ allocated in Slot } s \\ 0, \text{lecture of Teaching } t \text{ not allocated in Slot } s \end{cases}$$

[1] Comprehends both the Teachings that have to be allocated and those that have already been allocated and are considered as constraints for the model.

[2] I only consider the Teachers related to the Teachings to allocate.

As can be observed, the number of variables is lower when compared to those used in the previous theses, resulting in a lighter and faster model.

Functions

- **Teachings Slots**: a function $getSlots(t)$ returns the number of Slots in a Week for a Teaching:

$$getSlots : T \to \mathbb{N}$$

$$t \mapsto t_j \in \mathbb{N}$$

- **Get Slots in Day**: a function $getSlotsInDay(d)$ returns the list of Slots that belong to a certain Day d:

$$getSlotsInDay : D \to 2^S$$

$$d \mapsto S_d \subseteq S$$

- **Get correlated Teachings**: a function $getCorrelatedTeachings(t)$ returns the list of Teachings correlated with the Teaching t:

$$getCorrelatedTeachings : T \to 2^T$$

$$t \mapsto T_t \subseteq T$$

- **Get first Slot of a Day**: a function $getFirstSlotOfDay(d)$ returns the first Slot of the Day d:

$$getFirstSlotOfDay : D \to S$$

$$d \mapsto S_d \in S$$

- **Get last Slot of a Day**: a function $getLastSlotOfDay(d)$ returns the last Slot of the Day d:

$$getLastSlotOfDay : D \to S$$

$$d \mapsto S_d \in S$$

- **Get first Slot of a Day**: a function $getFirstSlotOfDayTeaching(t, d)$ returns the first lecture Slot of the Day d for the Teaching t:

$$getFirstSlotOfDayTeaching : T \times D \to S$$

$$t,\ d \mapsto S_{t,d} \in S$$

- **Get last Slot of a Day**: a function $getLastSlotOfDayTeaching(t, d)$ returns the last lecture Slot of the Day d for the Teaching t:

$$getLastSlotOfDayTeaching : T \times D \to S$$

$$t,\ d \mapsto S_{t,d} \in S$$

- **Get Teacher's Teachings**: a function $getTeacherTeachings(i)$ returns the Teachings taught by the Teacher i:

$$getTeacherTeachings : I \to 2^T$$

$$i \mapsto T_i \subseteq T$$

- **Get Teacher's unavailable Slots**: a function $getUnavailableSlots(i)$ returns the Slots in which the Teacher i is unavailable:

$$getUnavailableSlots : I \to 2^S$$

$$i \mapsto S_i \subseteq S$$

- **Get Teacher's preferences respected**: a function $teacherPreferencesRespected(t)$ calculates the number of preferences expressed by the Teaching t's Main Teacher that have been respected in the allocation:

$$teacherPreferencesRespected : T \to \mathbb{N}$$

$$t \mapsto t_j \in \mathbb{N}$$

Constraints

Constraints on the rows

Here I provide an analysis of the constraints that apply to each row individually (note: each row represents a Teaching):

- **Lectures Slots in a Week**: The sum of the cells in one row (i.e., the number of weekly Slots of a Teaching) of the matrix should be equal to the number of weekly lecture Slots of that Teaching.

$$\forall t \in T,\ n\_slots := getSlots(t),\ \sum_{s \in S} timetable\_matrix_{t,s} = n\_slots$$

- **Double Slots**: Considering only the cells related to Slots that belong to the same Day, a maximum of two of these cells can be equal to 1 at the same time and, if there are two cells equal to 1, they should be consecutive (i.e. if there are two cells equal to 1 in the same Day, the sum of each pair (slot, slot+1) of that Day should be either 0 or 2).

$$\forall d \in D,\ \forall t \in T,\ \sum_{s \in getSlotsInDay(d)} timetable\_matrix_{t,s} \leq 2$$

$$\forall d \in D,\ \forall s \in [1, getSlotsInDay(d) - 1],\ \forall t \in T,$$
$$timetable\_matrix_{t,s} + timetable\_matrix_{t,s+1} \in \{0, 2\}$$

- **Teachers' unavailable Slots**: The sum of the Slots in a Teaching where a Teacher is unavailable should be 0.

$$\forall i \in I,\ t \in getTeacherTeachings(i),\ unSlots := getUnavailableSlots(i)$$

$$\sum_{s \in unSlots} timetable\_matrix_{t,s} = 0$$

- **Teachers' preferences**: The sum of the Slots in a Teaching that respect the Main Teacher's preferences should be maximized.

$$\forall t \in T,\ \text{maximize } teacherPreferencesRespected(t)$$

Constraints on the columns

I now describe the constraints that apply to each column of the matrix (note: each column represents one time Slot):

- **Overlaps - Students**: Considering two correlated Teachings, the sum of the cells that belong to the same columns for those two Teachings should be ≤ 1, in order to avoid overlaps between them.

$$\forall s \in S,\ \forall t \in T,\ \sum_{t_{corr} \in getCorrelatedTeachings(t)} timetable\_matrix_{t_{corr},s} \in \{0, 1\}$$

- **Overlaps - Teachers**: Teachings taught by the same Teacher should not overlap.

$$\forall i \in I,\ t_{teacher} := getTeacherTeachings(i),\ s \in S,$$
$$\sum_{t \in t_{teacher}} timetable\_matrix_{t,s} \in \{0, 1\}$$

Constraints both on the rows and the columns

Finally, there are some constraints that, in order to be expressed, must consider both the rows and the columns of the matrix:

- **Maximum number of lectures in a Day**: The sum of correlated Teachings in a Day should be under a threshold.

$$\forall d \in D,\ \forall t \in T,$$

$$\sum_{\substack{t_{corr} \in getCorrelatedTeachings(t),\\ s \in getSlotsInDay(d)}} timetable\_matrix_{t_{corr},s} \leq \text{threshold}$$

- **Limit correlation between first and last Slot**: Considering two correlated Teachings, the sum of the cell that represents the first lecture Slot of the Day of one Teaching and the cell that represents the last lecture Slot of the same Day of the other Teaching should be ≤ 1.

$$\forall d \in D,\ \forall t \in T,\ \forall t_{corr} \in getCorrelatedTeachings(t),$$

$$s_f := getFirstSlotOfDay(d),\ s_l := getLastSlotOfDay(d)$$

$$timetable\_matrix_{t,s_f} + timetable\_matrix_{t_{corr},s_l} \in \{0, 1\}$$

- **Minimize lecture dispersion**: Considering two correlated Teachings, the distance between the first cell of the Day that is = 1 of one Teaching and the last cell of the same Day that is = 1 of the other Teaching should be minimized.

$$\forall d \in D,\ \forall t \in T,\ \forall t_{corr} \in getCorrelatedTeachings(t),$$

$$s_f := getFirstSlotOfDayTeaching(t, d),$$

$$s_l := getLastSlotOfDayTeaching(t_{corr}, d),$$

$$\text{minimize } timetable\_matrix_{t_{corr},s_l} - timetable\_matrix_{t,s_f}$$

Conclusions

In order to test the performances of this model, I use data related to the academic year 2023/24[3]. The results are encouraging: the Slot Model used in the previous

[3]I use the data related to the academic year 2023/24 because the model used in the previous thesis was run using this data, so I have a direct comparison.

thesis could find a solution in 36 hours[4], while the Bidimensional Matrix Model finds a solution in around 3 hours[5].

However, this model presents some criticalities:

- It cannot take classrooms into account, since adding this constraint would be too complex.
- The model works well with a medium amount of Teachings (around 320), but when the number of Teachings is large (> 700) it does not scale well.
- The soft constraints might represent a problem: if they are too many, CPLEX tries to optimize the solution indefinitely and does not provide any results.

[4]Using a server with 64 threads and 100GB of RAM.

[5]Using a PC with 16 threads and 32GB of RAM.

Course ID	Mon 8.30	Mon 10.00	Mon 11.30	Mon 13.00	Mon 14.30	Mon 16.00	Mon 17.30	Tue 8.30	Tue 10.00	Tue 11.30	Tue 13.00	Tue 14.30	Tue 16.00	Tue 17.30	Wed 8.30	Wed 10.00
293178	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
295768	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
295769	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
295844	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
294577	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0
294578	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0
294579	1	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0
294578	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0
293833	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
295823	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0
295098	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
293411	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
295093	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
294836	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
295822	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
295583	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.7: Example of a portion of the Bidimensional matrix

Chapter 5

Implementation

Finding the optimal way to implement the algorithm that generates the timetable is a process that requires a lot of effort, due to difficulties related to the Teachings' organization in the different Degree Courses, especially Computer Engineering. This chapter provides an analysis of those problems in detail, together with an explanation of how they have been solved.

5.1 Retrieving Courses data

Before implementing the algorithm to generate the timetables, a critical aspect that must be taken into consideration is the one related to retrieving the Courses data. As said in Chapter 4, those data are stored in different Excel files. Therefore, the information has to be first extracted from those files and saved in the database. I retrieve information about:

- The Degree Courses in ICM and ETF Colleges, their Orientations, and the Teachings in each Orientation.
- The type of each Teaching (Mandatory, Mandatory - Chosen from table, Free credit) and the correlations between them.
- The Teachers of a Teaching (both the Main Teacher and the collaborators).
- The number of Theory lectures hours and preference about their allocation.
- The organization of Practice and Laboratory Slots (number of Groups, number of Slots per Week, preference about Slot organization).
- The Slots in which the Teachers are unavailable.

5.2 Incremental approach

In an initial phase of the thesis, I tried to generate a timetable with all the Degree Courses from the a.y. 2023/24 together. Using this approach, the time needed to find a solution is around 36 hours, and the timetable that is generated is not optimal since the number of lectures in a Day is too high and there are too many consecutive empty Slots.

The main problems are due to the relations between Degree Courses, Computer Engineering and Mechatronic Engineering in particular. These two Degree Courses have many Teachings and Teachers in common, which causes difficulties in generating a timetable. CPLEX either fails to find a solution or returns suboptimal results after an extended computation.

For this reason, I explored an alternative approach: instead of generating a timetable by considering all the courses simultaneously, I try to generate many smaller timetables, each with a subset of Degree Courses. This is done by keeping the Teachings that had already been assigned to a timetable fixed and incrementally adding the other Teachings on top of them.
This methodology allows having different parameters for each subset of courses. For example, when generating the timetable for Computer Engineering, I can set the maximum number of correlations in a Day to 700 and, when generating the timetable for Mechatronic Engineering, I can change this number to 800, which was not possible when considering all the courses together.

This new approach drastically improves performance, reducing the time needed to find a solution to just 3 hours. Above this, I am also able to add constraints that were not considered previously, such as the Teacher's preferences about lecture Slots organization, and refine the parameters to reduce the number of lectures in a Day and of consecutive empty Slots.

The final order of Degree Courses is the following:

- *Ingegneria Informatica (Computer Engineering).*
- All other Degree Courses except *Ingegneria Informatica*, *Mechatronic Engineering (Ingegneria Meccatronica)*, and *Ingegneria Elettronica.*
- *Mechatronic Engineering (Ingegneria Meccatronica).*
- *Ingegneria Informatica.*
- *Ingegneria Elettronica.*

```
{
    "courses": [
        "INGEGNERIA DEL CINEMA E DEI MEZZI DI COMUNICAZIONE",
        "CYBERSECURITY",
        "DATA SCIENCE AND ENGINEERING",

        "ELECTRONIC AND COMMUNICATIONS ENGINEERING (INGEGNERIA ELETTRONICA E DELLE COMUNICAZIONI)",
        "INGEGNERIA FISICA",
        "AGRITECH ENGINEERING",
        "COMMUNICATIONS ENGINEERING",
        "ICT FOR SMART SOCIETIES (ICT PER LA SOCIETA' DEL FUTURO)",
        "NANOTECHNOLOGIES FOR ICTs (NANOTECNOLOGIE PER LE ICT)",
        "PHYSICS OF COMPLEX SYSTEMS(FISICA DEI SISTEMI COMPLESSI)",
        "QUANTUM ENGINEERING",
        "INGEGNERIA ELETTRONICA (ELECTRONIC ENGINEERING)"
    ],
    "orientations": [],
    "course_type": "",
    "max_corr_in_day": 800,
    "max_corr_first_last_slot": 20,
    "min_corr_overlaps": 35,
    "no_overlap_mandatory_practice_lab": False,
    "no_overlap_groups": False,
    "teachers_unavailabilities": True
},
```

Figure 5.1: Example of subset of Degree Courses with the parameters

5.3 Computer Engineering

Generating a timetable for Computer Engineering requires particular attention. This is due to the large number of its free credits, no other Degree Course has as many (for example, the Software Orientation alone includes 18 Teachings in the first semester of the second year). This means that generating a timetable is a challenging task, since all those Teachings must be considered.

When trying to generate a timetable with all the Degree Courses simultaneously, CPLEX struggles to find a solution. However, by excluding Computer Engineering, the solution is found in 2 hours and 15 minutes.

Therefore, using the incremental approach, first of all I generate a timetable for Computer Engineering only and then add the other Degree Courses. This allows generating the Computer Engineering timetable in about 50 minutes.

5.4 2026 Teachings

After generating and validating a timetable for the courses related to the academic year 2023/24, I can consider the courses for the academic year 2025/26. This represents a challenge as well, since the number of Teachings in 2025/26 is more than double that of 2023/24[1]. This means that it is not possible to find a solution using the same parameters as for the 2023/24 data. A review of the parameters and the constraints is therefore needed.

In this case, the main problem is related to the correlations between Teachings; the correlation values used in the previous theses could not be applied to the Teachings of the academic year 2025/26. Therefore, I conduct some tests to find correlation values that ensure no overlaps between main courses while still guaranteeing a feasible solution.

The conclusion is that the problem is related to the correlation values for the Teachings that can be chosen from a table. Therefore, I change the correlation for Teachings chosen from a table from 90 to

$$100/n\_teachings\_in\_table$$

while maintaining the same correlation for mandatory Teachings and free credits as before. I then refine the parameters according to these new correlations.

5.5 Teachers preferences about lectures organization

When considering soft constraints, one that requires attention is the Teachers' preferences about the Slots organization of their lecture. For each Teaching, the Main Teacher can express a preference on how they want the Theory, Practice, and Laboratory lecture Slots to be organized. For example, a Teacher can specify that the Laboratory lectures for a Teaching should be allocated in two Slots per Week, and that those Slots should be consecutive.

Adding these constraints represents a challenge. It not only requires the redefinition of the parameters, since the timetables generated change due to the different allocation of the lecture Slots, but it also introduces a new problem as well: since the number of soft constraints is large, CPLEX has difficulties in handling them, and when it tries to optimize the objective function, it cannot reach the objective prefixed and continues to run indefinitely.

[1]This is due to the fact that the Degree Courses offered by the Politecnico di Torino change every year and, generally, the number of Teachings increases over time

For this reason, it is necessary to introduce a limit to the execution time: after 6 hours, CPLEX returns the best solution it could find for a subset of Degree Courses, even if it is not the most optimized one.

ORGANIZZAZIONE_BLOCCHI_LEZIONE	ORGANIZZAZIONE_BLOCCHI_ESERCITAZIONE	ORGANIZZAZIONE_BLOCCHI_LAIB_ATENEO
tutti i blocchi da 3h	tutti i blocchi da 1,5h per ciascuna squadra	blocchi da 3h per ciascuna squadra
tutti i blocchi da 3h	tutti i blocchi da 1,5h per ciascuna squadra	blocchi da 3h per ciascuna squadra
tutti i blocchi da 3h	tutti i blocchi da 3h per ciascuna squadra	blocchi da 3h per ciascuna squadra
tutti i blocchi da 3h	tutti i blocchi da 3h per ciascuna squadra	blocchi da 1,5h per ciascuna squadra
un blocco da 3h e gli altri da 1,5h	un blocco da 3h e gli altri da 1,5h per ciascuna squadra	blocchi da 1,5h per ciascuna squadra
un blocco da 3h e gli altri da 1,5h	un blocco da 3h e gli altri da 1,5h per ciascuna squadra	blocchi da 1,5h per ciascuna squadra
tutti i blocchi da 3h	tutti i blocchi da 3h per ciascuna squadra	indifferente

Figure 5.2: Teachers' preferences (they are in Italian because they are saved in this format in the university's office)

5.6 ICM, ETF, and All Courses Timetables

When reviewing the timetables, I discovered that many problems arise from the relationships between the ETF and ICM Degree Courses. These two Colleges share a large number of Teachings, as well as Teachers. Therefore, in order to be able to generate a timetable that takes both Colleges into account, I must allow overlaps between the Practices and Laboratories of mandatory Teachings and those of Teachings chosen from a table, as well as overlaps between different Groups of Practice or Laboratory of the same Teaching.

Instead, by considering ICM and ETF College separately, generating timetables is much simpler and I do not have to make such compromises.

In the final version of the project, you will find three different timetables: one that considers ICM Courses only, one with ETF Courses only, and one with the

Courses from both of those Colleges combined.

5.7 Technical debt management: the usage of SonarQube

Another main challenge during this thesis is to write clean, understandable code that can be easily reused and modified. This is a process that should not be overlooked, since understanding the strategies used for the timetable generation and how they have been implemented is a very difficult task, but it is necessary for someone who wants to modify and improve the algorithm.

Therefore, to better manage the code, reduce technical debt, and keep its complexity to a minimum, I use the SonarQube software. This tool connects to a Git repository, analyzes the code every time it is pushed to that repository, and produces a report about the code quality. This report highlights security, reliability, and maintainability issues. Solving these issues means less technical debt and better code quality and readability. Therefore, part of the time is spent on code refactoring in order to reduce the number of issues raised by SonarQube as much as possible. When working on those issues, I start from the high-priority ones (as classified by SonarQube) and, once those have been solved, I can focus on medium and low-priority issues.

The final number of issues is 17, all related to maintainability. I have to accept those 17 issues in order to have more readable code, solving them would make the code too complex to analyze.

Figure 5.3: SonarQube Code Analysis

Chapter 6

How to use the tool

The thesis is divided into three distinct projects: *Excel_to_db_converter*, *Timetable_Allocator*, and *GUI_orario_Tesi*. This chapter describes how to use those three tools and their parameters. This is aimed at both those who simply want to use the tools to generate timetables and those who want to modify the code to add or remove constraints. Please refer to the *README* file of the project for more details.

6.1 Excel_to_db_converter

As described in Chapter 4, most of the information about the Teachings and the Teachers (number of Theory, Practice, and Laboratory hours, Teachers' preferences about the Slot allocation, and Teachers' unavailabilities) is saved in Excel files. This Python script retrieves the data about Teachers and Teachings from those Excel files and saves it in the database so that it can be used by the Allocator.

To run the program, simply execute the main.py file.

First, the script retrieves the information about Degree Courses, Orientations, and Teachings from the file *Percorsi-gruppi-insegnamenti aa 2026.xlsx*.

Once this information has been retrieved, the script reads the Excel files in the *Courses Data* folder and the *PreferenzeDocenti.xlsx* file and saves the data related to each Teaching and Teacher in the Database.

In the Utils folder, there are 3 files:

- **Teaching.py**: This file contains a class that represents a Teaching. The list of Teachings was previously loaded from the file *Percorsi-gruppi-insegnamenti aa 2026.xlsx* and therefore is retrieved from the database.
- **Get_Teachings_Data.py**: This file contains the functions that retrieve the information about the Teachings from the Excel files and save it in the

Database.

- **Get_Teachers_Data.py**: This file contains the functions that retrieve the information about the Teachers (Slots in which they are unavailable and the preferences about the allocation of their Teachings).

```
'''Teachings'''
# Get the Degree Courses related to DAUIN and DET departments (IDs CL003 and CL006)
get_teachings()
# Calculate the correlation for each Teaching
calculate_correlations()

# Load the Teachings from the DB
list_teachings = db_api.get_teachings()
teachings = []

# Converting the data in the list
for row in list_teachings:
    teachings.append(Teaching(id_teaching=row[0], title=row[1], main_teacher=row[2]))

# Get the number of lecture hours from the Excel files and insert it in the database
get_teaching_information(teachings)


'''Teachers'''
# Get the Teachers preferences for the courses and save them in the database
get_teachers_preferences(teachings)

get_practice_lab_not_in_preferences()

# Get the information about unavailable Slots for each Teacher from the PreferenzeDocenti.xlsx Excel file
# and insert them in the database
get_teachers_unavailabilities()
```

Listing 6.1: Retrieving information about Teachings and Teachers

6.2 Timetable_Allocator

Once all the data needed is loaded by the *Excel_to_DB_Converter* and inserted in the Database, the user can utilize the *Timetable_Allocator* project to generate a Timetable.

First of all, the script initializes the Teachings and Teachers classes, retrieving the information from the Database, and then it generates a timetable using the incremental approach described in Chapter 5.

There are different aspects of the *Timetable_Allocator* that require attention when using or modifying it. They are listed here.

6.2.1 Start from existing timetable

When executing the script, users can decide if they want to use an existing timetable from previous years as a starting point to generate the new one.

The program asks the user if they want to use this methodology and, if their reply is affirmative, the program reads the database table PreviousSolution and loads the data related to that timetable in the CPLEX software. Then, CPLEX generates a solution using that timetable as a starting point.

This has two benefits: it ensures a faster execution time, since CPLEX does not have to start from scratch but tries to optimize a timetable that already exists, and it will generate a solution that is similar to the one provided.

However, if the new constraints differ significantly from those used when generating the previous timetable, CPLEX will abandon the optimization of the existing timetable and will ignore it.

6.2.2 Allocation on Saturday

The software provides the possibility to allocate Slots on Saturday.

When executing the script, the user is asked whether to enable allocation on Saturday. If enabled, the user must specify the number of Slots that can be allocated on Saturday (minimum 1, maximum 7). Then, the timetable is generated, and Teachings can be allocated on Saturday the same way they can be allocated on other Days.

6.2.3 Export to Excel

After the timetable is generated, the user is given the option to export it to an Excel file. If they want to do so, two files are generated and saved in the folder Data/*timetable_name*.

The first Excel file contains the timetables saved in weekly format: each sheet corresponds to a Degree Course and contains separate timetables for every Orientation and Year. The timetables are represented as tables with the Days in the columns and the hours in the rows.

Orientation: Cryptography expert - Year: 1-1		
index	**Lun**	**Mar**
8.30-10.00	Big data: architectures and data analytics (Lezione) Quantum computing (Lezione)	Quantum computing (Lezione)
10.00-11.30	Computer architectures and operating systems (Lezione)	Big data: architectures and data analytics (Laboratorio - Squadra2)
11.30-13.00	Optimization methods and algorithms (Lezione)	Information systems security (Esercitazione - Squadra2) Big data: architectures and data analytics (Laboratorio - Squadra1)
13.00-14.30	Information systems security (Lezione)	Information systems security (Esercitazione - Squadra1)
14.30-16.00	Networks & Cloud Technologies and Security (Lezione)	Networks & Cloud Technologies and Security (Lezione)
16.00-17.30	Machine learning for networking (Lezione) Quantum computing (Lezione)	Information systems security (Lezione)
17.30-19.00	Information systems security (Laboratorio - Squadra2)	Information systems security (Laboratorio - Squadra2)

Figure 6.1: Timetable with weekly visualization

The second Excel file instead focuses on the single Teachings: there is the same structure with one sheet for each Degree Course, and for each of them the timetables are divided by Orientation and Year. But, in this case, there is one row for each Teaching, correlated with its Slots. This visualization is suitable for a Teacher who wants to view the Slots for their Teachings.

Orientation: Cryptography expert - Year: 2-1	
Teaching	**Slots**
Advanced cryptography	Lun 10.00-11.30 (Esercitazione - Squadra1) Lun 13.00-14.30 (Lezione) Ven 11.30-13.00 (Lezione)
AI and Cybersecurity	Gio 11.30-13.00 (Lezione) Ven 10.00-11.30 (Lezione)
Cybersecurity laws and regulations	Mer 14.30-16.00 (Lezione) Gio 14.30-16.00 (Lezione) Ven 13.00-14.30 (Lezione)
Data protection, Privacy and Anonymity	Lun 11.30-13.00 (Lezione) Ven 14.30-16.00 (Lezione)
Security verification and testing	Lun 8.30-10.00 (Lezione) Mer 16.00-17.30 (Laboratorio - Squadra1) Mer 17.30-19.00 (Laboratorio - Squadra1) Gio 10.00-11.30 (Lezione) Gio 16.00-17.30 (Laboratorio - Squadra2) Gio 17.30-19.00 (Laboratorio - Squadra2) Ven 17.30-19.00 (Lezione)
Advanced Information systems security	Lun 14.30-16.00 (Esercitazione - Squadra1) Lun 16.00-17.30 (Laboratorio - Squadra1) Lun 17.30-19.00 (Laboratorio - Squadra1) Mar 14.30-16.00 (Lezione) Mar 16.00-17.30 (Laboratorio - Squadra2) Mar 17.30-19.00 (Laboratorio - Squadra2) Mer 11.30-13.00 (Lezione) Mer 16.00-17.30 (Laboratorio - Squadra2) Mer 17.30-19.00 (Laboratorio - Squadra2) Gio 16.00-17.30 (Laboratorio - Squadra1) Gio 17.30-19.00 (Laboratorio - Squadra1) Ven 8.30-10.00 (Lezione)

Figure 6.2: Timetable with Teaching visualization

6.2.4 Function add_teachings_constraints

This function adds the constraints related to the Teachings. For each individual constraint, a dedicated function is called, where the constraint is defined and added to the model. If the constraint must also apply to Practices and/or Laboratories, an additional function is called to add the constraint to those lecture types.

```
'''
    Add the constraint about the number of Slots that each
   Teaching should have in a Week
'''
def add_slots_per_week_teaching(model, timetable_matrix, teachings
   , slots):
    for teaching in teachings:
        model.add_constraint(model.sum(timetable_matrix[teaching.
   id_teaching, s] for s in slots) == teaching.lect_slots)

        '''Practice Slots'''
        add_slots_per_week_practice(model, timetable_matrix,
   teaching, slots)

        '''Lab Slots'''
        add_slots_per_week_lab(model, timetable_matrix, teaching,
   slots)
```

Listing 6.2: Example of a function that adds a Teaching constraint

6.2.5 add_teachers_constraints

Similarly to the function *add_teachings_constraints*, this function adds the constraints related to the Teachers.

```
'''
    Teachings with the same Teacher should not overlap
'''
def add_no_overlap_constraint(model, timetable_matrix, teacher,
   slots):
    # Getting the IDs of the Teachings taught by "teacher" for the
    first and the second semester
    teaching_ids_1, teaching_ids_2 = get_teaching_ids(teacher)

    for s in slots:
        # Constraints for the first semester
        model.add(model.sum(timetable_matrix[t_id, s] for t_id in
   teaching_ids_1) <= 1)

        # Constraints for the second semester
        model.add(model.sum(timetable_matrix[t_id, s] for t_id in
   teaching_ids_2) <= 1)
```

Listing 6.3: Example of a function that adds a Teaching constraint

6.2.6 Parameters

The Allocator uses a series of parameters that can be adjusted to obtain a better timetable or to reduce the time needed to generate one. These parameters are defined in the file Parameters.py.

- **course_order**: This structure defines the sets of Degree Courses used to generate the timetable. Changing the order of these Degree Courses can change the time needed to generate a timetable as well as the results. Along with the each set of Degree Courses, it is possible to change the parameters used to generate the timetable for that set.
- **slot_per_day**: number of lecture Slots per Day. Default is 7.
- **n_weeks_in_semester**: number of Weeks in a Semester, used to calculate how many Slots per Week need to be allocated to a Teaching. Default is 14.
- **hours_in_slot**: number of hours in a Slot. At the moment, Slots are 1.5 hours long.
- **start_from_previous_solution**: boolean variable that is true if starting from an existing solution is enabled. Default is false.
- **saturday_enabled**: boolean variable that is true if lecture allocation on Saturday is enabled. Default is false.
- **n_slots_saturday**: number of Slots on Saturday. Minimum is 1, maximum is 7, default is 4.
- **max_corr_in_day**: number of maximum correlated lectures in a Day.
- **max_corr_first_last_slot**: maximum correlation value between the first and last Slot of the Day. The lower the number, the least student will have lecture at 8:30 and 17:30 in the same Day.
- **min_corr_overlaps**: the minimum number of correlation between Teachings for which it is guaranteed that there are no overlaps.
- **no_overlap_mandatory_practice_lab**: if true, Practices and Laboratories of mandatory Teachings cannot overlap with lectures of other correlated Teachings.
- **no_overlap_groups**: if true, groups of the same Practice or Laboratory cannot overlap with each other.
- **teachers_unavailabilities**: if false, the algorithm does not consider Teachers' unavailabilities when generating the timetable.

- **max_consecutive_slots_teaching**: maximum number of consecutive Slots that a Teaching can have.
- **teaching_overlap_penalty**: the penalty in the objective function for overlaps between Teaching with a correlation $<$ *min_corr_overlaps*.
- **lecture_dispersion_penalty**: the penalty in the objective function for lecture dispersion (defined as the difference between the first and last lecture Slot of a Day).
- **teacher_preferences_penalty**: the penalty in the objective function for Teachers' preferences about Slot organization that have not been respected (note that this value should be negative, since the objective is to maximize the number of Teachers' preferences respected).
- **consecutive_groups_penalty**: the penalty in the objective function for different Practice or Laboratories groups of the same Day that are not consecutive (note that this value should be negative, since the objective is to maximize the number of consecutive groups).
- **timetable_name**: the name with which the timetable is saved in the DB.
- **days** and **time_slots**: the name of the Days ("Lun", "Mar", "Mer", etc. - in Italian because they are saved like this in the Database) and the Slots ("8.30-10.00", "10.00-11.30", etc.), used when saving the timetable in the DB.

6.3 Web Application

During the previous theses, in order to be able to visualize the timetables generated and easily read them, a web application that displays the timetable via a GUI was developed. This web application reads the generated timetable in the Database and displays it in a graphical, user-friendly way. In this thesis, the web application has been extended and new functionalities have been added.

6.3.1 Allocation plan section

The web application is organized in different sections. The most important one when validating the timetables is the one related to the allocation plan. Here users can select the Allocation Plan that they are interested in validating and visualize the weekly timetable, divided by the degree type (Bachelor's or Master's Degree), the Degree Course, the Semester, and the Orientation.

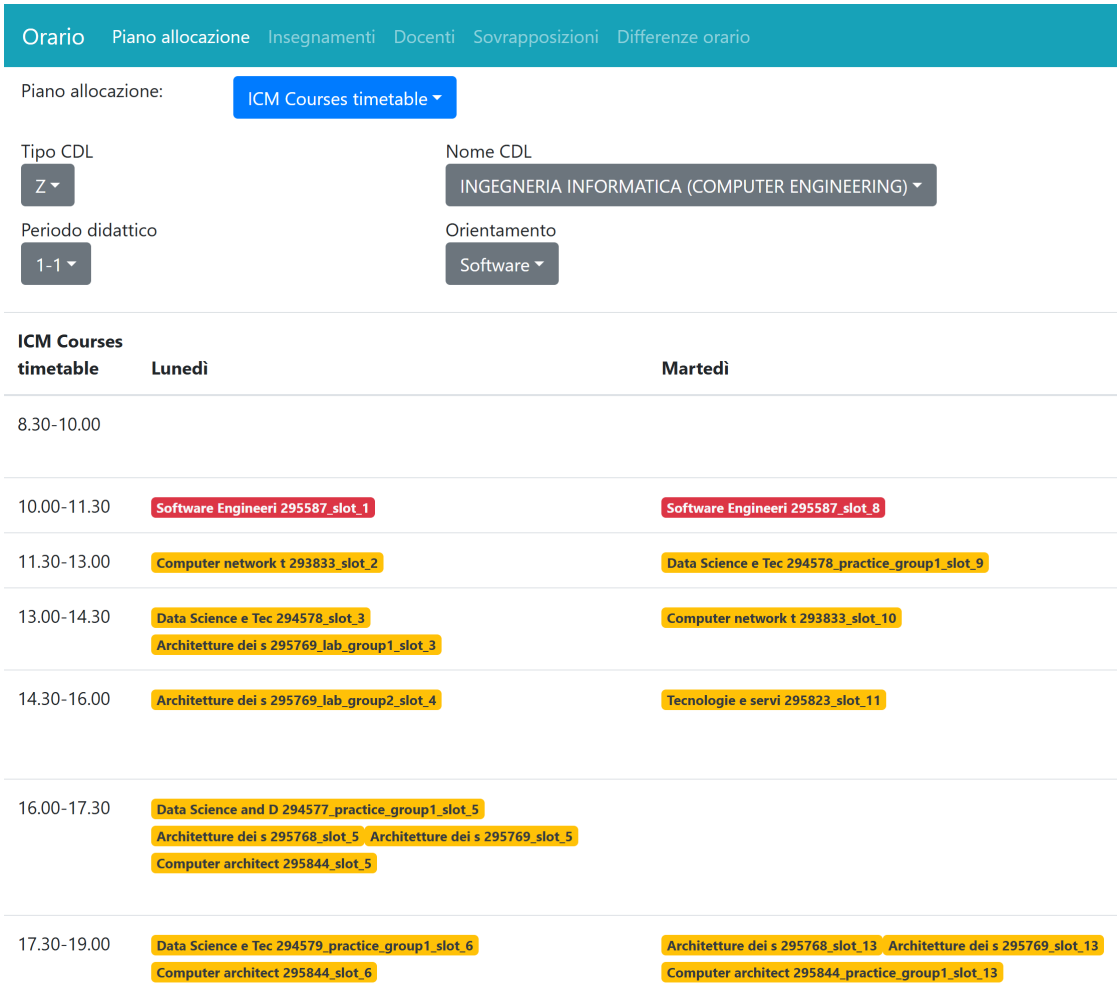

Figure 6.3: Allocation plan section - Teachings in red are mandatory and Teachings in yellow are chosen from a table

6.3.2 Teachers section

The Teachers section allows users to visualize the Slots for each Teacher and analyze if they have too many consecutive hours or not. Users can select the allocation plan and the Teacher they are interested in and see its Teachings.

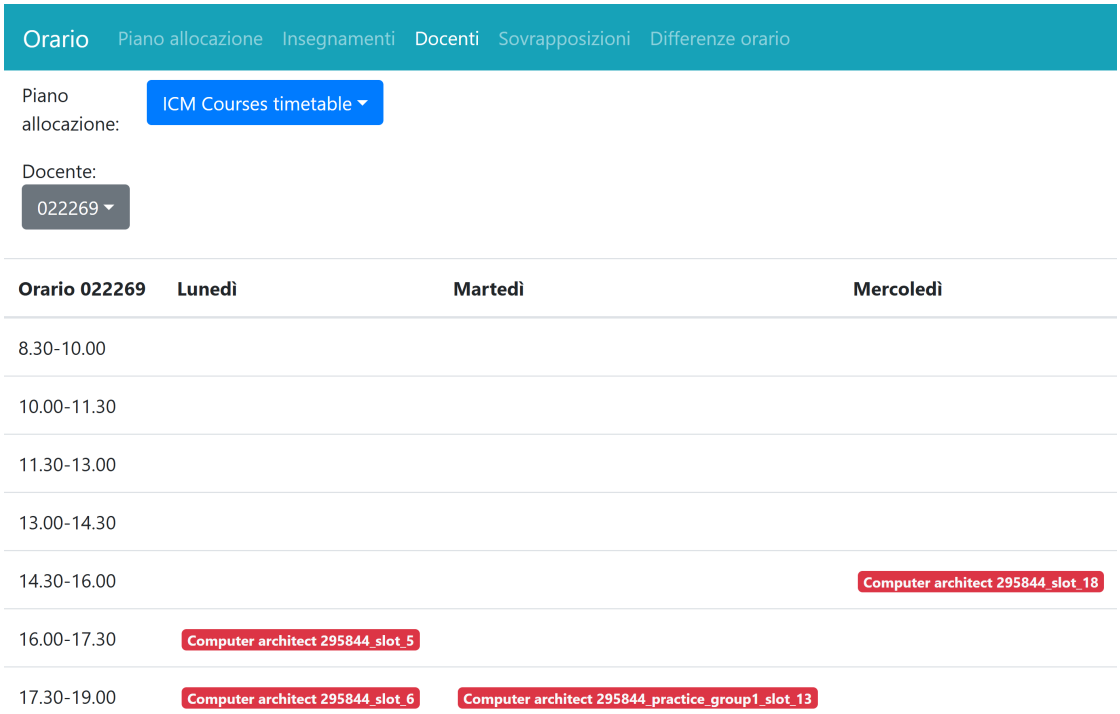

Figure 6.4: Teachers section

6.3.3 Timetable differences section

A new section has been added during this thesis: the Timetable differences section. This section is useful to compare the timetables generated with another timetable (for example, the timetables generated during this thesis were compared with the one generated during the previous works relating to the academic year 2023/24) in order to visualize the differences in the allocation of the single Teachings.

This section also provides an analysis that quantifies the number of Days and Slots that differ between the two timetables and how they have changed (if the Teaching's Slots have been moved to the beginning or the end of the Week or whether they remain unchanged).

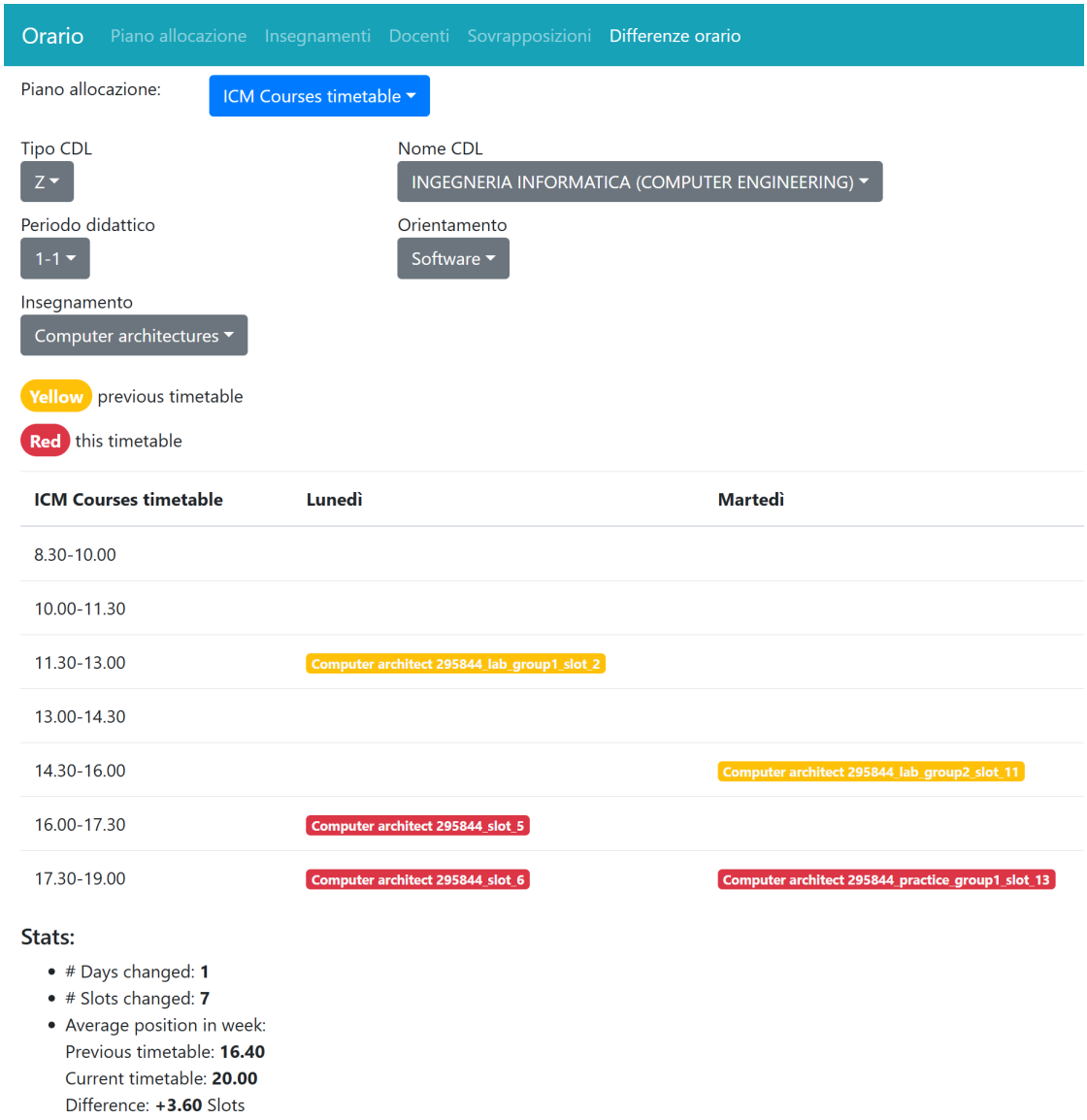

Figure 6.5: Timetable differences section

Chapter 7

Validation and performances

After generating a timetable, it has to be validated in order to check its goodness and that it can be used in a real-case scenario. This chapter analyzes the validation process and the performance of the model. Please note that all the performances are referred to a PC with a 16-thread processor and 32GB of RAM.

7.1 Checking timetable goodness - Students

One of the most important analyses that have to be made in order to check the validity of a timetable is the one related to the Students' constraints. For this validation, I check that the numbers of consecutive Teachings and empty Slots that Students have in a Day are limited and that the overlaps between Teachings are not critical. In order to do so, I visualize the timetable using the web application.

This is a trial-and-error process: I generate a timetable, check its validity, change the constraints and parameters to solve the problems that emerge, and re-generate the timetable.

7.2 Checking timetable goodness - Teachers

Along with constraints related to the Students, another important part of the timetable is represented by the constraints related to the Teachers. Using the web application, I am able to check if a Teacher has too many consecutive lecture Slots and if the Slots in which the Teacher is unavailable have been respected, and adjust the constraints consequently in case of problems.

Since the constraints about Teachers are less than those about Students, this process is simpler.

7.3 Analyzing differences with previous timetable

After solving the problems related to the constraints, I am interested in analyzing the differences between the timetables generated in this work and those generated in the previous theses. I can perform this analysis by using the *Timetable differences* section of the web application.

The conclusion is that the algorithm developed in this thesis explores the space of possible solutions in a different way compared to the one implemented in the previous thesis, leading to timetables that are dissimilar from those generated by the previous algorithm. For the majority of the Teachings, the number of Slots changed is high (above 80%). But, it is also worth mentioning that the changes on Days are few, with zero or one Days changed compared to the previous timetable.

Another interesting analysis is about the Teaching's average position in the Week. It is calculated as

$$\frac{\sum slot\_position}{\#slots\_in\_week}$$

and is needed in order to study if a Teaching is towards the start of the Week (low values), the end of the Week (high values), or the middle. I then compared the average position of the timetables generated in this thesis with the previous ones and concluded that some of the Teachings have been moved towards the end of the Week, some have been moved towards the start, and some have maintained a similar position.

7.4 Performances with data from the academic year 2023/24

As mentioned before, in order to validate the new Bidimensional Matrix Model implemented, in an initial phase I generate timetables using the data related to the academic year 2023/24.

7.4.1 Mechatronic Engineering

I start the tests by considering Mechatronic Engineering only, so that the time needed to generate the timetables (and check the validity of the constraints implemented) is minimal. Considering only hard constraints, the algorithm is able to generate timetables in less than a minute. This time increases to 10 minutes when I introduce soft constraints.

7.4.2 All courses

After the timetables for Mechatronic Engineering are good and I am satisfied with the hard and soft constraints, I extend the algorithm and consider all the Degree Courses related to the academic year 2023/24.

Using the Bidimensional Matrix model together with the incremental approach described in Chapter 5, I obtain a good performance, finding an admissible solution in less than 10 minutes and generating an optimal solution in around 3 hours (compared to the 36 hours needed with the model used in the previous thesis). However, this solution does not consider the Teachers' preferences about Slot organization, which are introduced later with the data for the academic year 2025/26.

7.5 Performances with data for the academic year 2025/26

Once the data related to the academic year 2025/26 is available in the form of an Excel file, I add that data to the database and generate the timetables with these Degree Courses.

In this case, since there are more Teachings, the generation is longer. The algorithm is able to find an admissible solution in 1 hour and 30 minutes and generate an optimized solution in 4 hours and 30 minutes, not considering the Teachers' preferences.

7.5.1 Teachers' preferences

The final constraint added is about the Teachers' preferences on the organization of the Slots for their Teachings. This constraint introduces new problems, since there are too many soft constraints and therefore CPLEX tries to optimize the timetable indefinitely and never returns a solution.

For this reason, I have to introduce a limit on CPLEX's execution time, ensuring that the software returns a solution within a maximum of 6 hours for each subset of Degree Courses. This means that the time needed to find a solution increases to 13 hours.

7.6 Questionnaire for Teachers and Student's Representatives

After generating the timetable for the 2025/26 Degree Courses, I prepared two questionnaires, one for the Teachers and the other for the Students' representatives,

in order to ask them for their opinion about the timetable and if there are any criticalities or improvements that can be made.

When answering the questionnaire, Teachers and Students are provided with the timetable in the form of an Excel file and are asked the following questions:

- If they are satisfied by the timetable.
- If the distribution of lectures in a Week is adequate.
- If there are overlaps that can represent a problem.
- If the lecture Slots and empty Slots are well distributed.
- If the timetable generated encourages the attendace.
- If the timetable allows for good time management between lectures and study/other commitments.
- If the preferences about Slots organizations have been respected.
- A comparison between the timetable generated and the official one.

One of the main criticalities highlighted by the questionnaire concerns the fact that some Teachings do not have a sufficient number of Slots during the Week. After double-checking the information related to these Teachings, it emerged that, for some of them, the approximations made when converting the total number of hours for a semester into weekly Slots resulted in an insufficient allocation. A review of these data is therefore required.

Another issue is the dissimilarity between the timetable generated in this thesis and the official one, as reported in Figure 7.1. For this reason, studying how CPLEX explores the space of possible solutions and having it return results that are in line with the existing timetable can be beneficial.

As highlighted in Figure 7.2, a third important aspect that emerged when analysing the questionnaire's results is about the Teachers' preferences. Many Teachers report that the preferences expressed about the allocation of their Teachings have only been respected partially or have not been respected at all. This therefore requires a redefinition of the soft constraint related to the Teachers' preferences, assigning a higher penalty if they are not respected.

On a positive note, as shown in Figure 7.3, the Lecture distribution during the Week seems balanced, and the majority of the people who answered the questionnaire say that there is a good division of time between lectures, research, studying, and other commitments.

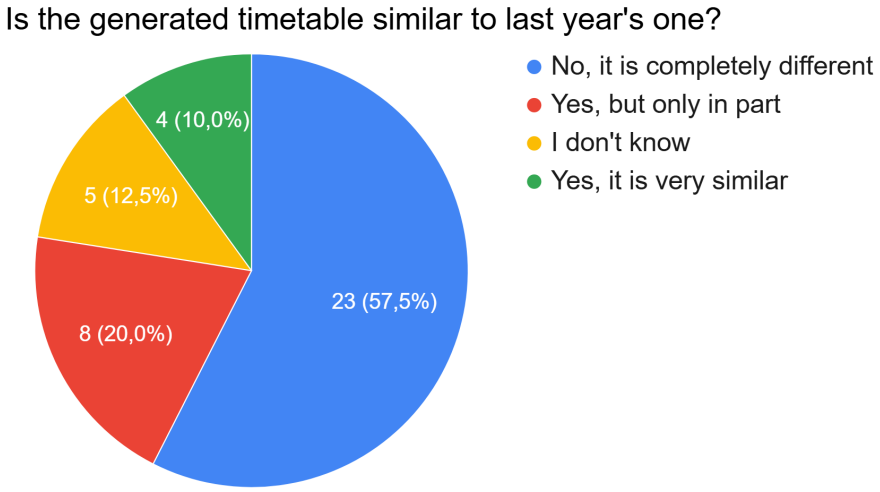

Figure 7.1: Comparison between the timetable generated and the last year's one

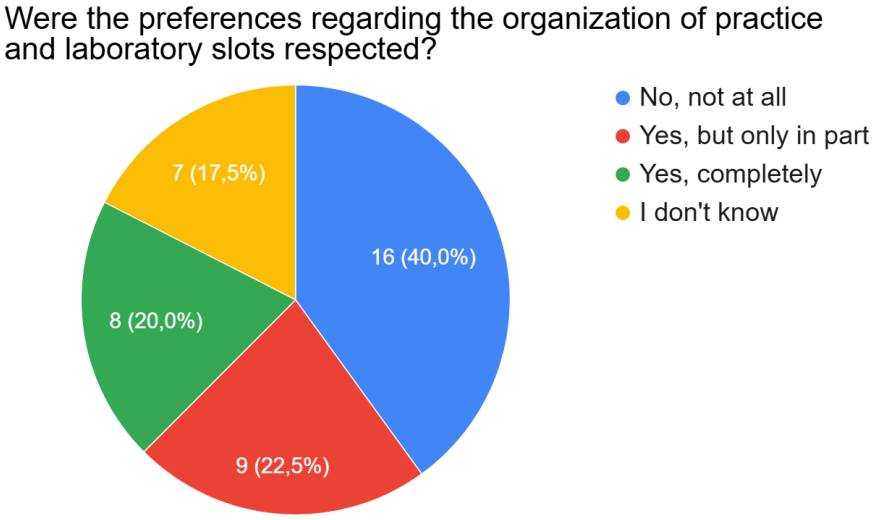

Figure 7.2: Teachers preferences on Slots allocation

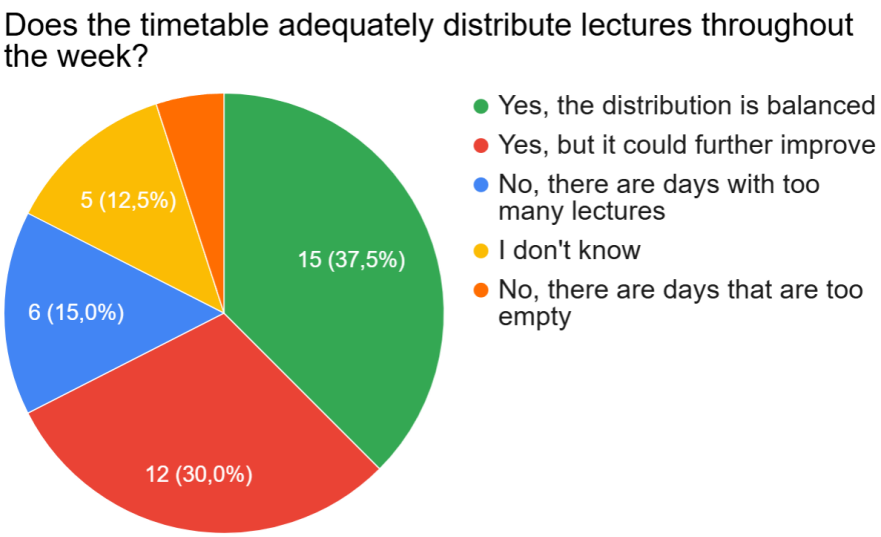

Figure 7.3: Lecture distribution over the week

Chapter 8

Conclusions

The timetable generation is a challenging task, due to the amount of Teachings involved, the fact that the list of those Teachings evolves every year, and the number of constraints that the timetable must satisfy. Therefore, some approximations had to be made and some constraints had to be relaxed in order to be able to generate a solution. Finding the right approximations that make it possible to generate a solution while maintaining a good representation of reality is one of the major problems encountered during this thesis, and the one on which I spent the most amount of time.

Although I had to face these problems, the results are encouraging: the algorithm is able to generate timetables in a few hours (compared to days in the previous theses) and those generated seem to be balanced, with a good lecture distribution over the Week.

The further development of the correlation model studied in the previous thesis, and its combination with the Bidimensional Matrix Model and the incremental approach, allows the algorithm to generate solutions in a more efficient way, while always maintaining a good representation of the possible Teachings combinations that Students can choose. The Bidimensional Matrix Model is in fact capable of generating timetables faster than the Slot Model used in the previous thesis, where timetable generations could take up to 400% more time.

The questionnaire sent to the Teachers and the Students' representatives is important to receive feedback from people directly affected by the timetable, as well as highlighting problems with the timetable generated.
A strength point of the algorithm is that the timetables that it generates are balanced, with a good distribution of Teachings during the Week.
However, some critical aspects emerged from the questionnaire, mainly related to the Teachers preferences about Slots allocation and the data regarding the Teachings, on which I based the timetable generation. A review of these aspects will be important in future works, in order to solve these issues and obtain a

timetable that better satisfies Teachers and Students needs.

Another potential problem with the timetable generated during this thesis can be the differences when compared to the one generated in the previous work. This algorithm explores the space of possible solutions in a different way, leading to timetables that can be profoundly different when compared to the official one or those from the previous theses.

Finally, the CPLEX software proved to work well with a medium number of Teachings (around 300) and constraints, but when this number increases and more constraints are introduced, it does not scale well, leading to higher execution times and difficulties in finding a solution.

8.1 Future works

The final section of this thesis gives some ideas for possible future works.

An interesting analysis that can be done is related to the method used to generate the timetables: in this thesis, I used ILP with CPLEX in order to explore the new Bidimensional Matrix approach, but since CPLEX does not scale well when there are too many Teachings and Constraints, it might be worth exploring different methodologies to represent and solve the problem, that do not rely on CPLEX or ILP.
In recent years, new optimization algorithms with equality and inequality constraints have been studied. In particular, in [20] and [21], the authors propose a novel methodology, called Controlled Multipliers Optimization, which starts from the Lagrangian of the problem and defines a dynamic system where a control law is applied to the Lagrange multipliers. This approach makes it possible to reach a minimum of the optimization problem with a lower computational time compared to classical algorithms, such as interior-point methods. Moreover, this method can effectively solve mixed-integer problems, such as the timetable one, which makes it an interesting approach; see [20] for more details.

Other possible future works are related to the application of the Genetic Algorithm and the Ant Colony System algorithm analyzed in Chapter 4 to solve the timetable problem. They can be a valid alternative to ILP for this kind of problems, even though, as discussed, representing more complex constraints might be challenging.

Lastly, the FET software analyzed in Chapter 4 can be further studied, and its performance can be compared to the algorithm implemented in this thesis.

Possible improvements of this work include:

- Avoiding CPLEX running indefinitely when optimizing the solution, requiring a time limit.

- Reviewing the data related to each Teaching, assigning the right amount of weekly Slots.
- Reviewing the penalties assigned to each soft constraint, generating timetables that are more in line with Teachers preferences.
- Attempting to generate timetables that are more similar to those generated in the previous theses or to the official one.
- Uniforming the tables and columns names in the database, in order to use the same standard in all of them.

Bibliography

[1] Francesco Lasagno. *Modellazione e ottimizzazione della pianificazione della didattica*. 2022. URL: https://webthesis.biblio.polito.it/23530/ (cit. on pp. 1, 11, 13, 14, 35).

[2] Manuel Messina. *Nuovi metodi di modellazione e ottimizzazione per la pianificazione della didattica*. 2023. URL: https://webthesis.biblio.polito.it/27660/ (cit. on pp. 1, 11, 13, 14, 35).

[3] Bob Stanke. *Operations Research: Optimizing Decision-Making for Success*. July 2023. URL: https://www.bobstanke.com/blog/operations-research-overview (cit. on p. 5).

[4] IBM. *Tutorial: Linear Programming*. URL: https://ibmdecisionoptimization.github.io/tutorials/html/Linear_Programming.html (cit. on p. 5).

[5] IBM. *Optimization Software*. URL: https://www.ibm.com/optimization-solver (cit. on p. 6).

[6] CPLEX User's Manual. «Ibm ilog cplex optimization studio». In: *Version* 12.1987-2018 (1987), p. 1 (cit. on p. 7).

[7] Solvice. *Constraints*. URL: https://www.solvice.io/glossary/constraints (cit. on p. 7).

[8] Geeks for Geeks. *Objective Function*. July 2025. URL: https://www.geeksforgeeks.org/maths/objective-function/ (cit. on p. 8).

[9] L. De Giovanni. *Note su Programmazione Lineare e Metodo del Simplesso*. URL: https://www.math.unipd.it/~luigi/courses/ricop/m02.PLsim.01.pdf (cit. on p. 8).

[10] Antonio Vetrò. *Technical Debt*. private communication. Nov. 2024 (cit. on p. 9).

[11] Wikipedia contributors. *SonarQube — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-July-2025]. 2024. URL: https://en.wikipedia.org/w/index.php?title=SonarQube&oldid=1263144878 (cit. on p. 9).

[12] Jumoke Soyemi, John Akinode, and Samson Oloruntoba. «Electronic Lecture Time-Table Scheduler Using Genetic Algorithm». In: *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/-DataCom/CyberSciTech)*. 2017, pp. 710–715. DOI: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.124 (cit. on p. 24).

[13] Mathworks. *What Is the Genetic Algorithm?* URL: https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html (cit. on p. 24).

[14] Tejo Herianto. «Implementation of the Ant Colony System Algorithm in the Lecture Scheduling Process». In: *Instal: Jurnal Komputer* 12.02 (2020), pp. 50–56 (cit. on p. 26).

[15] Geeks for Geeks. *Introduction to Ant Colony Optimization*. URL: https://www.geeksforgeeks.org/machine-learning/introduction-to-ant-colony-optimization/ (cit. on p. 27).

[16] Liviu Lalescu. *FET Free Timetabling Software*. URL: https://lalescu.ro/liviu/fet/ (cit. on p. 28).

[17] Jordi Castro. «A computational evaluation of optimization solvers for CTA». In: *International Conference on Privacy in Statistical Databases*. Springer. 2012, pp. 11–21 (cit. on pp. 31, 32).

[18] Josef Jablonsk et al. «Benchmarks for current linear and mixed integer optimization solvers». In: *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis* 63.6 (2015), pp. 1923–1928 (cit. on pp. 32, 33).

[19] Ashutosh Shenoy. *MIP Solvers Unleashed: A Beginner's Guide to PuLP, CPLEX, Gurobi, Google OR-Tools, and Pyomo*. Apr. 2025. URL: https://medium.com/operations-research-bit/mip-solvers-unleashed-a-beginners-guide-to-pulp-cplex-gurobi-google-or-tools-and-pyomo-0150d4bd3999 (cit. on p. 33).

[20] V. Cerone, S. M. Fosson, S. Pirrera, and D. Regruto. «A new framework for constrained optimization via feedback control of Lagrange multipliers». In: *IEEE Transactions on Automatic Control* (2025), pp. 1–16. DOI: 10.1109/TAC.2025.3568651 (cit. on p. 68).

[21] V. Cerone, S. M. Fosson, S. Pirrera, and D. Regruto. «A feedback control approach to convex optimization with inequality constraints». In: *2024 IEEE 63rd Conference on Decision and Control (CDC)*. 2024, pp. 2538–2543. DOI: 10.1109/CDC56724.2024.10885825 (cit. on p. 68).