INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

**Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores**

# Examination Timetabling Automation using Hybrid Meta-heuristics

Miguel De Brito e Nunes

(Licenciado em Engenharia Informática e de Computadores)

Trabalho de projeto realizado para obtenção do grau
de Mestre em Engenharia Informática e de Computadores

Relatório de Progresso

Orientadores:
Artur Jorge Ferreira
Nuno Miguel da Costa de Sousa Leite

**Março de 2015**

# Contents

# List of Figures

# List of Tables

# List of Code Listings

# 1

# Introduction

Many people believe that glsai was created to imitate human behavior and the way humans think and act. Even though people are not wrong, AI was also created to solve problems that humans are unable to solve, or to solve them in a shorter time window, with a better solution. Humans may take days to find a solution, or may not find a solution that fits their needs. Optimization algorithms, that is, methods that seek to minimize or to maximize some criterion, may deliver a very good solution in minutes, hours or days, depending on how much time the human is willing to use in order to get a proper solution.

A concrete example of this type of problems is the creation of timetables. Timetables can be used for educational purposes, sports scheduling, transportation timetabling, among other applications. The timetabling problem consists in scheduling a set of events (e.g., exams, people, trains) to a specified set of time slots, while respecting a predefined set of rules. In some cases, the search space is so limited by the constraints that one is forced to relax them in order to find a solution.

## 1.1 Educational Timetabling Problems

This class of timetabling problems involve the scheduling of classes, lectures or exams on a school or university in a predefined set of time slots while satisfying a set of rules. Examples of rules are: a student can't be present in two classes at the same time, a student can't have two exams on the same day or even an exam must be scheduled before another.

Depending on the institution type and if we're scheduling classes/lectures or exams the timetabling problem is divided into three main types:

- Examination timetabling - consists on scheduling university course exams avoiding the overlap of exams containing students from the same course and spreading the exams as much as possible in the timetable.
- Course timetabling - consists on scheduling lectures considering the multiple university courses, avoiding the overlap of lectures with common students.
- School timetabling - consists on scheduling all classes in a school, avoiding the need of students being present at multiple classes at the same time.

In this project, the main focus is the examination timetabling problem.

The process of creating a timetable requires that the final solution follows a set of constraints. The type of constraint depends on the timetabling problem type and problem specifications. The constraints are divided in two groups: *hard constraints* and *soft constraints*. Hard constraints are a set of rules which must be followed in order to get a feasible solution. On the other hand, soft constraints represent the views of the different interested players (e.g., institution, students, nurses, train operators) on the resulting timetable. The satisfaction of this type of constraints is not mandatory as is for the case of the hard constraints. In the timetabling problem, the goal is usually to optimize a function with a weighted combination of the different soft constraints, while satisfying the set of hard constraints.

## 1.2 Objectives

This project's main objective is the production of a prototype application which serves as an examination timetabling generator tool. The problem at hand focus on the specifications introduced in the ITC 2007, *First track*, which includes 12 benchmark instances. In the ITC 2007 specification, the examination timetabling problem considers a set of periods, room assignment, and the existence of constraints analogous to the ones present in real instances.

The application's requirements are the following:

- Automated generation of examination timetables, considering the ITC 2007 specifications (*mandatory*).
- Validation (correction and quality) of a timetable provided by the user (*mandatory*).
- Graphical User Interface to allow the user to edit generated solutions and to optimize user's edited solutions (*optional*).

This project is divided into two main phases. The first phase consists on studying some techniques and solutions for this problem emphasizing meta-heuristics like: Genetic Algorithms (GA), Simulated Annealing (SA), Tabu Search (TS), and some of its hybridizations. The second phase consists in the development of the selected algorithms and promising hybridizations, and test them using ITC 2007 data. A performance comparison between the proposed algorithms and the state-of-the-art algorithms will be made.

## 1.3 Planning

In this section, more details are given about the development of the two project phases. A Gantt diagram is presented showing each project's phase timeline. On the first phase the author studied the educational timetabling problem and in particular the examination timetabling problem and related literature. This work is documented in this progress report. The second phase's main goal is to develop solutions for the ITC 2007 problem instance.

One of the approaches to be developed will use a local search algorithm (e.g., Simulated Annealing) to improve the solution obtained using a graph coloring heuristic. The results of this approach will be compared against other approaches applied to the ITC 2007's timetabling problem, namely the first five winners. In a subsequent approach, instead of using a single-solution based meta-heuristic, the author intends to use a population-based meta-heuristic (e.g., Genetic Algorithm) hybridized with a local search algorithm (e.g., Simulated Annealing), or a combination of other meta-heuristics. A performance comparison will be made in which the proposed solution methods are compared with the five ITC 2007 finalists and other state-of-the-art approaches.

As an optional project requirement, a GUI for editing generated solutions by the human planner will be developed.

The Gantt diagram presented on Figure 1.1 shows a detailed planning for the development of all the main topics of this project including both phases mentioned above. The colors presented on the diagram are green and blue, which designate the first and second phase, respectively.

## 1.4 Document Organization

The rest of the document is organized as follows. Sector 2, entitled State-of-the-Art, specifies the timetabling problem focusing on examination timetabling and existing approaches applied to the ITC 2007 benchmark data. This chapter includes a survey of the most important paradigms and algorithmic strategies used to tackle the timetabling problem. Next, the methods of the five ITC 2007 finalists are resumed. The next chapters should address the implementation aspects of the solution method. A chapter containing the evaluation of the proposed algorithms and comparison with other competing algorithms, and a final chapter containing the conclusions of the work, will also be included.

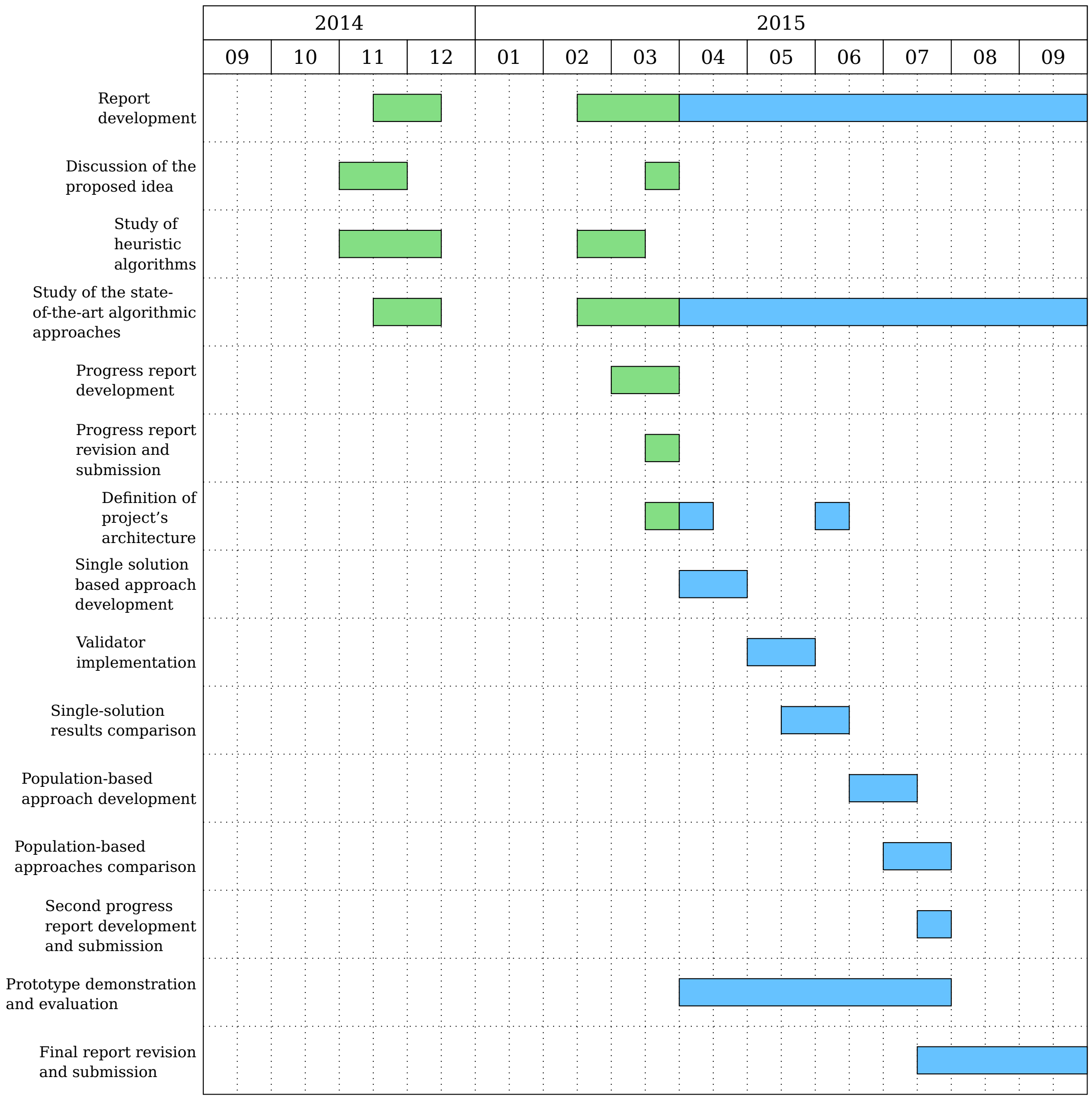

Figure 1.1: Gantt diagram of the project activities.

# 2

# State of the Art

In this section, we review the state-of-the-art of the problem at hand. We start by describing why timetabling is a rather complex problem, some possible approaches to solve it and some of the existing solutions, specifically for the ITC 2007 benchmarks.

## 2.1 Timetabling Problem

When solving timetabling problems, it is possible to generate one of multiple types of solutions which are *feasible*, *non feasible*, *optimal* or *sub-optimal*. A feasible solution solves all the mandatory constraints, unlike non feasible solutions. An optimal solution is the best possible feasible solution given the problem constraints. A problem may have multiple optimal solutions. Lastly, non-optimal solutions are feasible solutions that have sub-optimal values.

Timetabling automation is a subject that has been a target of research for about 50 years. The timetabling problem may be formulated as a search or an optimization problem [**?**]. As a search problem, the goal consists on finding a feasible solution that satisfies all the hard constraints, while ignoring the soft constraints. By posing the timetabling problem as an optimization problem, one seeks to minimize (considering a minimization problem) the violations of soft constraints while satisfying the hard constraints. Typically, the optimization is done after the use of a search procedure for finding an initial feasible solution.

The basic examination timetabling problem, where only the *clash* hard constraint is observed, reduces to the graph coloring problem [**?**], which is a well studied problem. The clash hard constraint specifies that no conflicting exams should be scheduled at the same time slot. Deciding whether a solution exists in the graph coloring problem, is a NP-complete problem [**?**]. Considering the graph coloring as an optimization problem, it is proven that the task of finding the optimal solution is also a NP-Hard problem [**?**]. Graph Coloring problems are explained further in Section 2.2

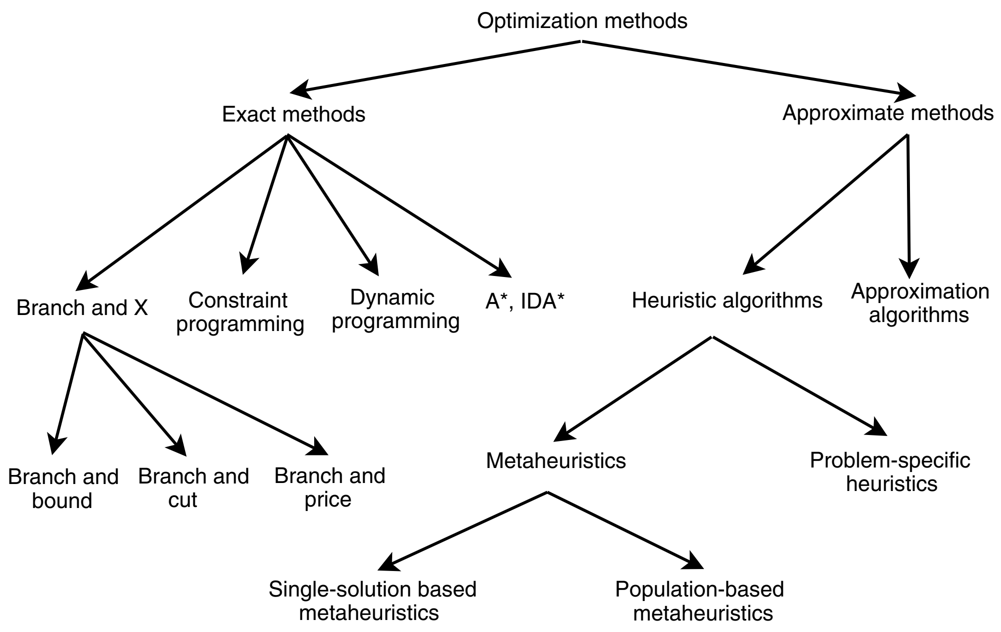

Figure 2.1: Optimization methods: taxonomy and organization (adapted from [**?**]).

## 2.2 Existing Approaches

Figure 2.1 depicts a taxonomy for the known optimization methods. These methods are divided into *Exact methods* and *Approximate methods*.

Timetabling solution approaches are usually divided into the following categories [**?**]: *exact algorithms* (Branch-and-Bound, Dynamic Programming), *graph based sequential techniques*, *local search based techniques* (Tabu Search, Simulated Annealing, Great Deluge), *population based algorithms* (Evolutionary Algorithms, Memetic algorithms, Ant Colony algorithms, Artificial immune algorithms), *Multi-criteria techniques*, *Hyper-heuristics*, *Decomposition/clustering techniques*. Hybrid algorithms, which combine features of several algorithms, comprise the state-of-the-art. Due to its complexity, approaching the examination timetabling problem using exact method approaches can only be done for small size instances. Real problem instances found in practice are usually of large size, making the use of exact methods impracticable. Heuristic solution algorithms have been usually employed to solve this problem.

Real problem instances are usually solved by applying algorithms which use both *heuristics* and *meta-heuristics*. Heuristic algorithms are problem-dependent, meaning that these are adapted to a specific problem in which one can take advantage of its details. Heuristics are used to generate a feasible solution, focusing on solving all hard constraints only. Meta-heuristics, on the other hand, are problem-independent. These are used to, given the feasible solution obtained using heuristic algorithms, generate a better solution focusing on satisfying as many soft constraints as possible.

Most of the existing meta-heuristic algorithms belong to one of the following three categories: One-Stage algorithms, Two-Stage algorithms and algorithms that allow relaxations [**?**]. The One-Stage algorithm is used to get a solution, which the goal is to satisfy both hard and soft constraints at the same time. The Two-Stage algorithms are the most used types of approaches. This category is divided in two phases: the first phase consists in not considering the soft constraints and focusing on solving hard constraints to obtain a feasible solution; the second phase is an attempt to find the best solution, trying to solve the largest number of soft constraints as possible, given the solution of the first phase. Algorithms that allow relaxation can weaken some constraints in order to solve the *relaxed problem*, while considering the elimination the used relaxations.

### 2.2.1 Exact methods

Approximation algorithms like heuristics and meta-heuristics proceed to enumerate partially the search space and, for that reason, they can't guarantee finding the optimal solution. Exact approaches perform a complete enumeration of the search space and thus guarantee that the encountered solution is optimal. A negative aspect is the time taken to find the solution. If the decision problem is very difficult (e.g. NP-Complete), in practical scenarios, given large size problem instances, using this approach may not be possible due to the long execution time.

#### Constraint-Programming Based Technique

The glscpbt allows direct programming with constraints which gives ease and flexibility in solving problems like timetabling. Two important features about this technique are the use of backtracking and logical variables. Constraint programming is different from other types of programming, as in these types it is specified the steps that need to be executed, but in constraint programming only the properties (hard constraints) of the solution, or the properties that should not be in the solution, are specified [**?**].

#### Integer Programming

The Integer Programming (IP) is a mathematical programming technique in which the optimization problem to be solved must be formulated as an Integer Problem. If the objective function and the constraints must be linear, and all problem variables are integer valued, then the IP problem is termed Integer Linear Problem (ILP). In the presence of both integer and continuous variables, then the problem is called Mixed-Integer Linear Programming (MILP). Schaerf [**?**] surveys some approaches using the MILP technique to school, course, and examination timetabling.

### 2.2.2 Graph Coloring Based Techniques

As mentioned previously, timetabling problems can be reduced to a graph coloring problem. Considering this relation between the two problems, several authors used two-phased algorithms in which graph coloring heuristics were applied in the first phase, to obtain an initial feasible solution.

#### Graph Coloring Problem

The Graph Coloring (GC) problem consists in assigning colors to an element type of a graph which must follow certain constraints. The simplest sub-type is the *vertex coloring*, which the main goal is to, given a number of vertices and edges, color the vertices so that no adjacent vertices have the same color. In this case the goal is to find a solution with the lowest number of colors as possible.

The examination timetabling problem can be transformed into a graph coloring problem as follows. The exams corresponds to vertices and there exists an edge connecting each pair of conflicting exams (exams that have students in common). With this mapping only the clash hard constraint is take into consideration. Thus, soft constraints are ignored [**?**].

Given the mapping between the GC problem and the examination timetabling problem, GC heuristics like *Saturation Degree Ordering* are very commonly used to get the initial solutions. Others like *First Fit* and other *Degree Based Ordering* techniques (*Largest Degree Ordering*, *Incidence Degree Ordering*) are also heuristic techniques for coloring graphs [**?**].

### 2.2.3 Meta-heuristics

Meta-heuristics, as mentioned above, usually provide solutions for optimization problems. In timetabling problems, meta-heuristic algorithms are used to optimize the feasible solutions provided by heuristics, like the GC heuristics. Meta-heuristics are divided in two main subtypes, which are *Single-solution meta-heuristics* and *Population-based meta-heuristics* [**?**].

#### Single-solution meta-heuristics

Single-solution meta-heuristics main goal is to modify and to optimize one single solution, maintaining the search focused on local regions. This type of meta-heuristic is therefore exploitation oriented. Some examples of this type are *SA*, *Variable-Neighborhood Search*, *TS*, and *Guided Local Search* [**?**].

#### Population-based meta-heuristics

Population-based meta-heuristics main goal is to modify and to optimize multiple candidate solutions, maintaining the search focused in the whole space. This type of meta-heuristic is therefore exploration oriented. Some examples of this type are *Particle Swarm*, *Evolutionary Algorithms*, and *Genetic Algorithms* [**?**].

### 2.2.4 ITC 2007 Examination timetabling problem: some approaches

In this section, the five ITC 2007 - Examination timetabling track - finalists approaches are described. This timetabling problem comprises 12 instances of different degree of complexity. Through the available website, competitors could submit their solutions for the given benchmark instances. Submitted solutions were evaluated as follows. First, it is checked if the solution is feasible and a so-called distance to the feasibility is computed. If it is feasible, the solution is further evaluated based on the fitness function, which measures the soft constraints total penalty. Then, competitors' solutions are ranked based on the distance to feasibility and solution's fitness value. The method achieving the lower distance to feasibility value is the winner. In the case of a tie, the competitor's solution with the lowest fitness value wins. A solution is considered feasible if the value of distance to feasibility is zero. The set of hard constraints is the following:

- no student must be elected to be present in more than one exam at the same time;
- the number of students in a class must not exceed the room's capacity;
- exam's length must not surpass the length of the assigned time slot;
- exams ordering hard constraints must be followed; e.g., $Exam_1$ must be scheduled after $Exam_2$;
- room assignments hard constraints must be followed; e.g., $Exam_1$ must be scheduled in $Room_1$.

It is also necessary to compute the fitness value of the solution which is calculated as an average sum of the soft constraints penalty. The soft constraints are listed below:

- two exams in a row - a student should not be assigned to be in two adjacent exams in the same day;
- two exams in a day - a student should not be assigned to be in two non adjacent exams in the same day;
- period spread - reduce the number of times a student is assigned to be in two exams that are $N$ time slots apart;
- mixed durations - reduce the number of exams with different durations that occur in the same room and period;
- larger exams constraints - reduce the number of large exams that occur later in the timetable;
- room penalty - avoid assigning exams to rooms with penalty;
- period penalty - avoid assigning exams to periods with penalty.

To get a detailed explanation on how to compute the values of fitness and distance to feasibility based on the weight of each constraint, please check the ITC 2007 website [**?**].

We now review the ITC 2007's five winners approaches. The winners list of the ITC 2007 competition is as follows:

- 1st Place - Tomáš Müller

- 2nd Place - Christos Gogos
- 3rd Place - Mitsunori Atsuta, Koji Nonobe, and Toshihide Ibaraki
- 4th Place - Geoffrey De Smet
- 5th Place - Nelishia Pillay

We now briefly describe these approaches.

Tomáš Müller's approach [**?**] was actually used to solve all three problems established by the ITC 2007 competition. He was able to win two of them and to be finalist on the third. For solving the problems, he opted for an hybrid approach, organized in a two-phase algorithm. In the first phase, Tomáš used the Iterative Forward Search (IFS) algorithm [**?**] to obtain feasible solutions and Conflict-based Statistics [**?**] to prevent IFS from looping. The second phase consists in using multiple optimization algorithms. These algorithms are applied in this order: Hill Climbing (HC) [**?**], Great Deluge (GD) [**?**], and optionally SA [**?**].

Gogos was able to reach second place in Examination Timetabling track, right after Müller. Gogos' approach [**?**], like Müller's, is a two-phase approach but with a pre-processing stage. The first phase starts by the application of a pre-processing stage, in which the hidden dependencies between exams are discovered in order to speed up the optimization phase. After the pre-processing stage, a construction stage takes place, using a meta-heuristic called *Greedy Randomized Adaptive Search Procedure (GRASP)*. In the second phase, optimization methods are applied in this order: HC, SA, IP (the Branch and Bound procedure), finishing with the so-called Shaking Stage, which is only used on certain conditions. Shaking Stage *shakes* the current solution creating an equally good solution, which is given to the SA. The objective of this stage to make SA restart with more promising solutions and generate better results.

Mitsunori Atsuta, Koji Nonobe and Toshihide Ibaraki ended up in third place on the Examination Timetabling track and won third and second place on the other tracks as well, with the same approach for all of them. The approach [**?**] consists on applying a constraint satisfaction problem solver adopting an hybridization of TS and Iterated Local Search.

Geoffrey De Smet's approach [**?**] differs from all others because he decided not to use a known problem-specific heuristic to obtain a feasible solution, but instead used what is called the *Drool's rule engine*, named *drools-solver* [**?**]. By definition, the drools-solver is a combination of optimization heuristics and meta-heuristics with very efficient score calculation. A solution's score is the sum of the weight of the constraints being broken. After obtaining a feasible solution, Geoffrey opted to use a local search algorithm to improve the solutions obtained using the drools-solver.

Nelishia Pillay opted to use a two-phase algorithm variation, using *Developmental Approach based on Cell Biology* [**?**], which the goal consists in forming a well-developed organism by the process of creating a cell, proceeding with cell division, cell interaction and cell migration. In this approach, each cell represents a time slot. The first phase represents the process of creating the first cell, cell division and cell interaction. The second phase represents the cell migration.

### 2.2.5 Other approaches

Salwani Abdullah, Hamza Turabieh, and Barry McCollum's 2009 approach [**?**] consists on using an hybridization of an electromagnetic-like mechanism and the GD algorithm. In this approach, the electromagnetism-like mechanism starts with a population of timetables randomly generated. Electromagnetic-like mechanism is a meta-heuristic algorithm using an *attraction-repulsion* mechanism [**?**] to move the solutions to the optimum.

Christos Gogos, George Goulas, Panayiotis Alefragis and Efthymios Housos' 2009 approach [**?**] also used the two-phase algorithm. The first phase consists on creating the timetable using a greedy randomized positioning, with the use of a backtracking mechanism to help creating the solution. This construction phase is repeated and only the best solutions pass to the second phase. The second phase utilizes local search (Isto não é um tipo?), SA, shaking and IP to improve the solutions. As mentioned by the author, this approach produced very good results and can be compared to Tomáš Müller's results in his approached used to win ITC 2007 competition.

B. McCollum, P.J. McMullan, A. J. Parkes, E.K. Burke and S. Abdullah's 2009 two phased approach [**?**] use an adaptive ordering heuristic from [**?**], proceeding with an *extended version* of GD. As the author stated, this approach is robust and general considering the results obtained on the benchmark datasets from ITC 2007 using this approach.

Malek Alzaqebah and Salwani Abdullah's 2011 two phased approach [**?**] starts by using a graph coloring heuristic (largest degree ordering) to generate the initial solution and ends by utilizing the *Artificial Bee Colony* search algorithm to optimize the solution.

Hamza Turabieh and SalwaniAbdullah's 2011 approach [**?**] utilize two phase algorithm. The first phase consists on constructing initial solutions by using an hybridization of graph coloring heuristics (least saturation degree, largest degree first and largest enrollment ? first). The second phase utilizes an hybridization of electromagnetic-like mechanism and GD algorithm, just like in 2009 Abdullah et al.'s approach.

Hamza Turabieh and Salwani Abdullah created another approach in 2012 [**?**]. It utilizes a Tabu-based memetic algorithm which consists on an hybridization of a genetic algorithm with a Tabu Search algorithm. The author states that this approach produces some of the best results when tested on ITC 2007's datasets.

P. Demeester, B. Bilgin, P. De Causmaecker and G. V. Berghe in 2012 created an hyper-heuristic approach [**?**]. The heuristics utilized are 'improved or equal' (hill climbing?), SA, GD and an adapted version of the *late acceptance* strategy [**?**]. These heuristics are used on already-created initial solutions. Initial solutions are constructed using an algorithm which does not guarantee the feasibility of the solution.

B. McCollum, P.J. McMullan, A. J. Parkes, E.K. Burke and S. Abdullah's 2012 approach [**?**] to ITC 2007 problem was based on IP formulation. This approach though, was not capable of solving the competition instances.

N. R. Sabar, M. Ayob, R. Qu and G. Kendall utilized a graphical coloring hyper-heuristic on its approach in 2012 [**?**]. This hyper-heuristic is composed of four hybridizations of these four methods: last degree, saturation degree, largest colored degree and largest enrollment. This approach seems to compete with the winners' approaches from ITC 2007, considering the benchmark results.

N. R. Sabar, M. Ayob, G. Kendall, R. Qub's 2012 approach [**?**] utilizes a two phase algorithm. Starts by using an hybridization of graph coloring heuristics to obtain feasible solutions and a variant of honey-bee algorithm for optimization. The hybridization is composed of least saturation degree, largest degree first, largest enrollment first applied in this order.

Salwani Abdullah and Malek Alzaqebah opted to create an hybridization approach in 2013 [**?**], mixing the utilization of a modified bees algorithm with local search algorithms (i.e. SA and late acceptance HC)

Salwani Abdullah and Malek Alzaqebah in 2014 constructed an approach [**?**] that utilizes an hybridization of a modified artificial bee colony with local search algorithm (i.e. late acceptance HC).

Edmund K. Burke, Rong Qu and Amr Soghier's 2014 approach [**?**] uses an hyper-heuristic with hybridization of low level heuristics (neighbor operations) to improve the solutions. The low level heuristics are *move exam*, *swap exam*, *kempe chain move* and *swap times lot*. After applying this hyper-heuristic with the hybridizations, the hybridization with the best results will be tested with multiple exam ordering methods, applying another hyper-heuristic with hybridizations. The heuristics applied are *largest degree*, *largest weighted degree*, *saturation degree*, *largest penalty* and *random ordering*.

Ali Hmer and Malek Mouhoub's approach [**?**] uses a multi-phase hybridization of meta-heuristics. Works like the two phase algorithm but includes a pre-processing phase before the construction phase. This pre-processing phase is divided in two phases: the propagation of ordering constraints and implicit constraints discovery. The construction phase utilizes a variant of TS. The optimization phase uses hybridization of HC, SA and an extended version of GD algorithm.

Ryan Hamilton-Bryce, Paul McMullan and Barry McCollum in 2014 opted to use a non-stochastic method on their approach [**?**] when choosing examinations in the neighborhood searching process on the optimization phase. Instead, it uses a technique to make an intelligent selection of examinations using information gathered in the construction phase. This approach is divided in 3 phases. The first phase uses a *Squeaky Wheel* constructor which generates multiple initial timetables and a weighted list for each timetable generated. Only the best timetable and its weighted list is passed to the second phase. The second phase is the *Directed Selection Optimization* phase which uses the weighted list created in the construction phase to influence the selection of examinations for the neighbor search process. ("DSO utilizes the weighted list to influence the selection of the examination for optimization".. ??? Não deveria ser seleção de exames para criar soluções vizinhas ao qual estas vão para optimização? Ou a melhor vai para optimização..). Only the best timetable is passed onto the next phase. The third phase is the *Highest Soft Constraint Optimization* phase which is similar to the previous phase but weighted list values are calculated based on the solution's individual soft constraints penalty.

S. A. Rahman, E. K. Burke, A. Bargiela, B. McCollum and E. Özcan's 2014 approach [**?**] is a constructive approach. This divides examinations in sets called *easy sets* and *hard sets*. Easy sets contain the examinations that are easy to schedule on a timetable and on the contrary, hard sets contain the ones that are hard to schedule and so are identified as the ones creating infeasibility. This allows to use the examinations present on the hard sets first on future construction attempts. There's also a sub-set within the easy set, called *Boundary Set* which helps on the examinations' ordering and shuffling. Initial examinations' ordering are accomplished by using graph coloring heuristics like largest degree and saturation degree heuristics.

Syariza A. Rahman, Andrzej Bargiela, Edmund K. Burke, Ender Özcan, Barry McCollum and Paul McMullan's 2014 approach [**?**] utilizes adaptive linear combinations of graph coloring heuristics (largest degree and saturation degree) with heuristic modifier. These adaptive linear combinations allows the attribution of difficulty scores to examinations depending on how hard their scheduling is. The ones with higher score, and so harder to schedule, are scheduled using two strategies: using single or multiple heuristics and with or without heuristic modifier. The authors conclude that multiple heuristics with heuristic modifier offers good quality solutions, and the presented adaptive linear combination is a highly effective method.

Nuno Leite, Fernando Melício and Agostinho C. Rosa's approach in 2014 [**?**] tries to solve not only the well-studied single epoch problem, but also an extension to that problem with two examination epochs. This approach utilizes a Memetic Algorithm to solve the problem. Memetic algorithms are hybridizations of a population-based meta-heuristic and a single-solution based meta-heuristic. The Memetic algorithm used is the hybridization of the *Shuffled Complex Evolution Algorithm* and the GD algorithm. Shuffled Complex Evolution Algorithm is a Evolutionary Algorithm. GD algorithm is a variant of the local search SA.

Table 2.1 Timeline

| | |
|---|---|
| 2007 | Atsuta et al. - Constraint satisfaction problem solver using an hybridization of TS and iterater local search |
| | Smet - drool’s solver with TS |
| | Pillay - Two phase approach using Developmental Approach based on Cell Biology, which the first phase consists on creating the first cell, cell division an cell iteration, and the second phase represents cell migrations |
| 2009 | Müller - Two-phase approach with hybridization, which the first phase includes Iterative Forward Search (IFS) and Conflict-based Statistics, and the second phase is composed of HC, GD and optionally SA |
| | Abdullah et al. - Hybridization of electromagnetic-like mechanism and GD algorithm |
| | Gogos et al. - Two phase approach, which the first phase consists on using a greedy randomized positioning with backtrack mechanism, and the second phase utilizes local search (?), SA, shaking and IP |
| | McCollum et al. - Two phase approach, which first phase consists on using adaptive ordering heuristic, and the second phase utilizes an *extended version* of GD |
| 2011 | Alzaqebah and Abdullah - Two phase approach, which the first phase uses the largest degree ordering, and the second phase utilizes an Artificial Bee Colony search algorithm |
| | Turabeih and Abdullah - Two phase approach, which the first phase utilizes an hybridization of graph coloring heuristics, and the second phase uses an hybridization of electromagnetic-like mechanism and GD algorithm |
| 2012 | Gogos - Considered a two-phase approach with a pre-processing stage for hidden dependencies. The first phase uses the *greedy randomized adaptive search procedure*, and the second phase is composed of HC, SA, IP with Branch and Bound, finishing with a shaking stage. |
| | Turabieh and Abdullah - Tabu-based memetic algorithm which is an hybridization of a genetic algorithm with TS |
| | Demeester et al. - Hyper-heuristic approach of ’improved or equal’, SA, GD and late acceptance strategy applied on already created solutions |
| | McCollum et al. - Approach based on IP formulation (problem was not solved) |
| | Sabar et al. - Graph coloring hyper-heuristic approach using last degree, saturation degree, largest colored degree and largest enrollment |
| | Sabar et al. - Two phase approach, which the first phase is composed of an hybridization of graph coloring heuristics and the second phase uses an honey-bee algorithm |
| 2013 | Abdullah and Alzaqebah - Hybridization approach with a modified bee algorithm and local search algorithms like SA and late acceptance HC |
| 2014 | Abdullah and Alzaqebah - Hybridization approach with a modified artificial bee colony with local search algorithm like late acceptance HC |
| | Burke et al. - Approach uses an hyper-heuristic with hybridization of low level heuristics (neighbor operations) like, thereafter uses an hyper-heuristic with hybridization of exam ordering methods |
| | Hmer and Mouhoub - Multi-phase hybridization of meta-heuristics. A two phase approach with pre-processing phase. First phase uses a variant of TS, and the second phase utilzies an hybridization of HC, SA and an extended version of GD algorithm |
| | Brice et al. - Approach with 3 phases. First phase uses a Squeaky Wheel constructor, second phase utilizes the weighted list created in the first phase for the neighbor search process, and the third phase uses a weighted list based on the solutions’ soft constraints penalty |
| | Rahman et al. - Constructive approach that divides examinations into easy and hard sets |
| | Rahman et al. - Approach utilizes adaptive linear combinations of graph coloring heuristics, like largest degree and saturation degree, with heuristic modifier |
| | Nuno Leite et al. - Memetic algorithm approach. Hybridization of Shuffled Complex Evolution algorithm and the GD algorithm |

# 3

# Architecture

This project is meant to participate in the ITC 2007. Comparing the approach and the results with the finalists of the competition is one of the main objectives of this project. Thus, it is ideal to develop an organized program with the best performance possible, so the implemented code can be easily understandable, extensible and so afford the best scores possible when creating timetable solutions. Taking that into account, this section is reserved to the description of the architecture and explanation of the most important decisions taken regarding this subject.

## 3.1 System Architecture

The architecture of this project is divided in multiple layers. These are independent from one another and each of them has its unique features considering the objectives of this project. The layers presented in the project are named *Data Layer (DL)*, *Data Access Layer (DAL)*, *Business Layer (BL)*, *Heuristics Layer (HL)*, *Tools Layer (TL)* and *Presentation Layer (PL)*.

The assortment, dependencies and the main classes of each layer can be seen in 3.1.

### 3.1.1 Data Layer

The DL is the layer where all entities are stored. Considering the fact that these entities must always be obtained from a file presented in ITC 2007's website [**?**], it does not make sense to keep these entities after the program runs. So, the entities are stored in volatile memory all the time and are discarded after the program is finished.

### 3.1.2 Data Access Layer

The DAL is the layer that allow access to the data stored in the DL and provide the signatures of all entities. This layer provides repositories of each type of entity. The repository was implemented in a way that it's signature is generic, and so can only be created with objects that implement IEntity. This is mandatory because the generic repository's implementation uses the identification presented in IEntity.

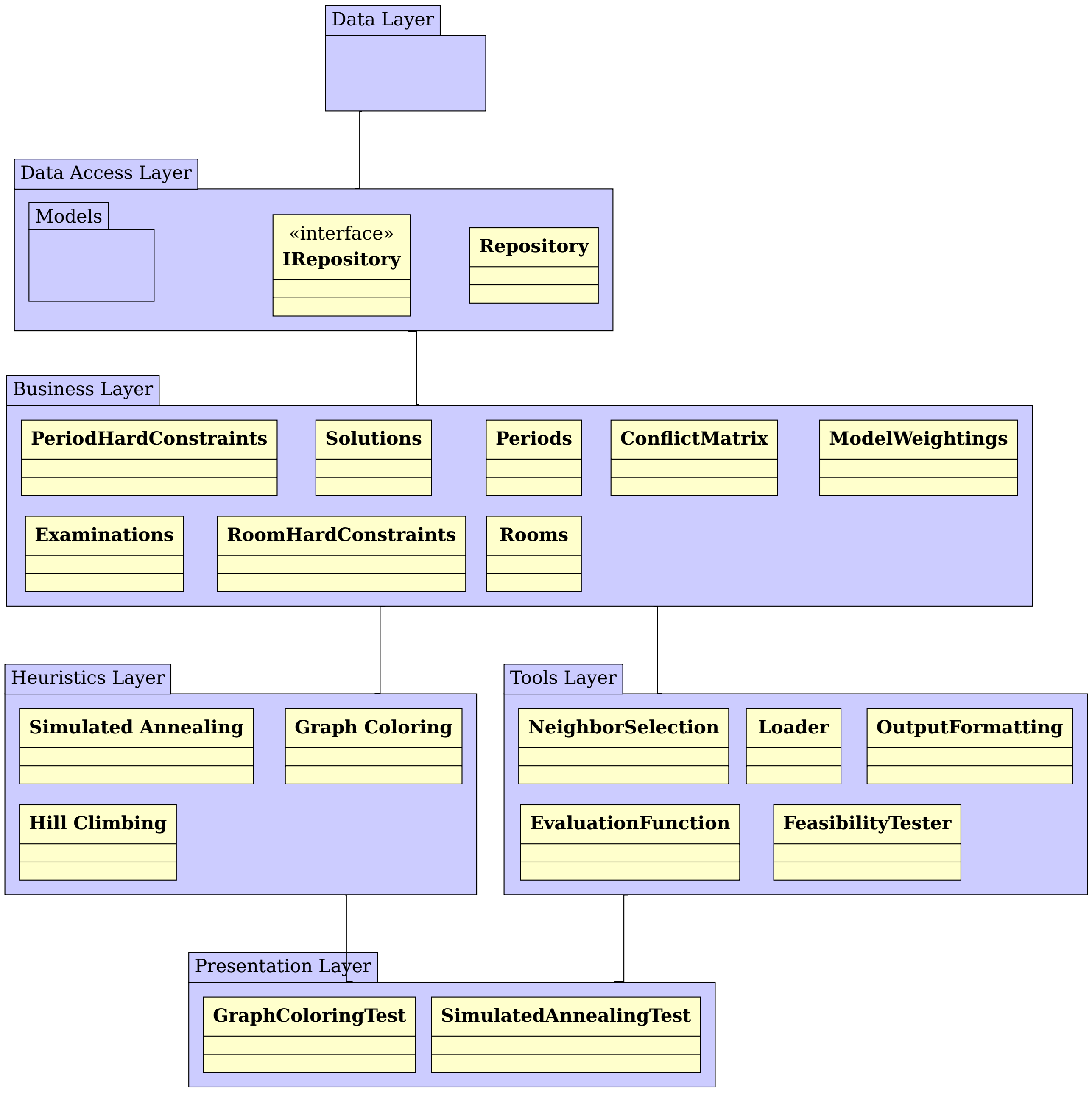

Figure 3.1: System Architecture

The Repository class stores the entities in a list, which index corresponds to the entity's identifier. The Repository provides basic CRUD functions to access and edit the entities stored. The signature of the entities and all the specifications of the generic repository can be seen in the 3.2.

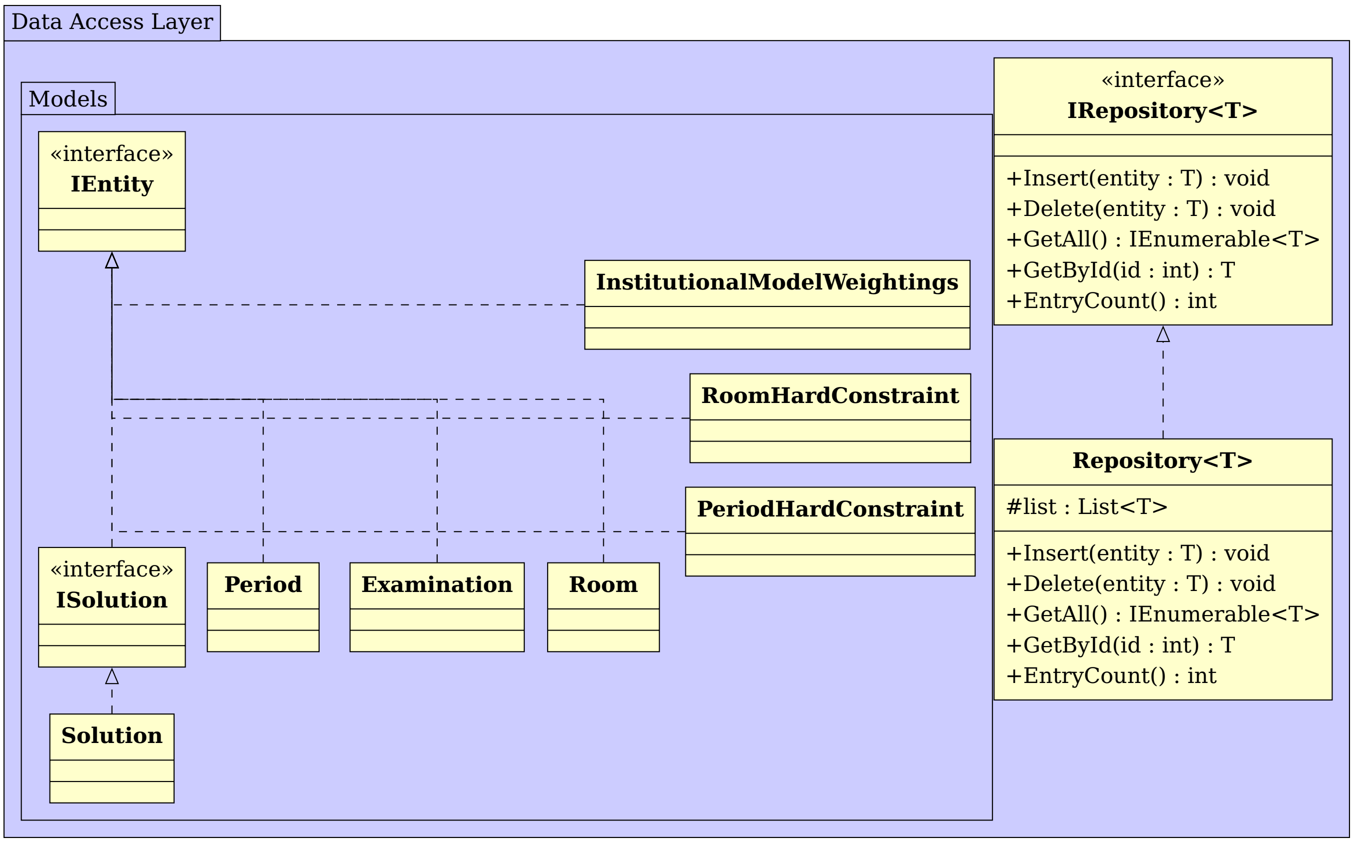

Figure 3.2: Data Access Layer

### 3.1.3 Business Layer

The BL provides access to the repositories explained above by, in each of the business classes, affording CRUD functions or get/set functions, and specific functions that depend on the type of the repository. One example of these last functions is, considering the room hard constraints repository, is to get all the room hard constraints that is of a certain type. This can be seen in the Figure 3.3, which includes all the BL classes, methods and variables. The CRUD functions provided by some of the business classes simply use the CRUD functions from the Repository instance, which is presented on that same business class.

Business classes afford CRUD functions if the objective is to store multiple instances of a certain type of entity. If that's the case, it uses a Repository instance of that type of entity. Thus, business classes that only store one instance, do not provide CRUD functions, just a *set* and *get* function. This makes sense, considering it only stores one instance of an entity, instead of multiple entity instances, not needing the use of a Repository. One example of these classes is the ConflictMatrix.

The ConflictMatrix provides the number of conflicts of each pair of examinations. Even though it's not an entity that implements IEntity, it must be easily accessed in the lower layers. There's only one instance of this class because there's only one conflict matrix for each set. Each set must be loaded using the Loader if that set is to be tested.

All the business classes implement the Singleton pattern. This decision was made because it makes sense to keep only one repository of each entity since they must be populated using the Loader each time a set is tested. Another reason is to avoid passing the instances references of the business classes to all the heuristics and tools that use them, and so, the instances can be easily be accessed statically.

### 3.1.4 Tools Layer

The TL represents the layer that contains all the tools used by the HL and by the lower layers, while using the BL to access all the stored entities. These tools are named EvaluationFunction, Loader, NeighborSelection, FeasibilityTester and OutputFormatting.

EvaluationFunction is a tool that allows the computation of the validation, fitness and distance to feasibility for a given solution. A solution is only valid if the examinations are all set, even if the solution is not feasible. The distance to feasibility determines the number of violated hard constraints and the fitness determines the score of the solution depending on the violated soft constraints and its penalty values. The distance to feasibility is used by the GC heuristic to guarantee that the end solution is feasible, while the fitness is used by the SA to compare the score of the current solution and the generated neighbors.

The Loader loads all the information presented in a file SET into the repositories. This tool is the first to be run allowing the heuristics and other tools to use the entities through the repositories. More information about this tool will be given in the section **X**.

The NeighborSelection is a tool that provides functions that verifies if a certain neighbor function can be applied in the current solution, if so, it returns a Neighbor object. A Neighbor object does not represent a neighbor solution, but the changes that need to be applied to the current solution if this neighbor is to be accepted. Details about this tool will be explained in the section **X**.

The FeasibilityTester is a tool that provides functions, which efficiently checks if a certain examination can be placed in a certain period, room or simultaneously period and room. This tool is used by both GC and SA, even though it only works if the examination to check is not yet set in the solution provided.

The OutputFormatting tool is used to create the output file given the final solution. This file obeys the output file rules represented in the ITC 2007's site [**?**] in order to be able to submit the solution [**?**]. Submitting the solution allows to check all violated hard constraints, soft constraints, distance to feasibility and fitness values on the site's page.

### 3.1.5 Heuristics Layer

The HL offers access to all the implemented heuristics. These are the GC, SA, and HC. All these heuristics are used to create the best timetable possible given a limited time. Heuristics like SA and HC make use of neighbor solutions, and so, utilize the NeighborSelection tool for this effect. They also use tools like FeasibilityTester and EvaluationFunction to help build the initial solution and check the fitness while improving the current solution, respectively.

Extensive explanation about these heuristics will be given in the section **X**.

### 3.1.6 Presentation Layer

The PL, in this phase, works mainly as a debugger to run the all the project functionalities and to check the final results. It's in this layer that all the tests are made, like checking execution timings and changing input parameters on SA and HC to check if better results can be achieved.

It is planned to be implemented by the final phase, another version of the presentation layer that includes the possibility of visualization of the final and best timetable generated.

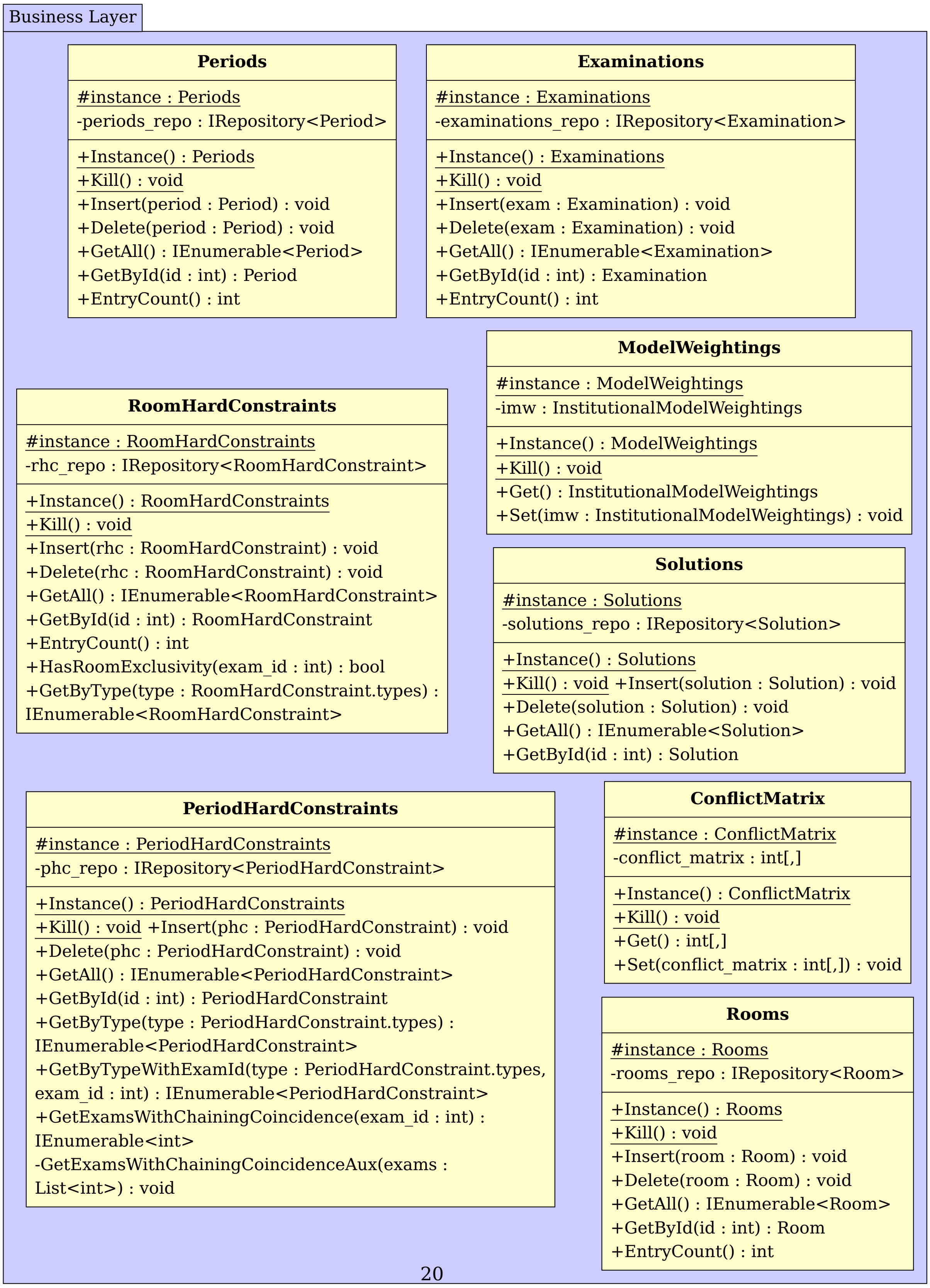
Business Layer
Periods
#instance : Periods
-periods_repo : IRepository<Period>
+Instance() : Periods
+Kill() : void
+Insert(period : Period) : void
+Delete(period : Period) : void
+GetAll() : IEnumerable<Period>
+GetById(id : int) : Period
+EntryCount() : int
Examinations
#instance : Examinations
-examinations_repo : IRepository<Examination>
+Instance() : Examinations
+Kill() : void
+Insert(exam : Examination) : void
+Delete(exam : Examination) : void
+GetAll() : IEnumerable<Examination>
+GetById(id : int) : Examination
+EntryCount() : int
ModelWeightings
#instance : ModelWeightings
-imw : InstitutionalModelWeightings
+Instance() : ModelWeightings
+Kill() : void
+Get() : InstitutionalModelWeightings
+Set(imw : InstitutionalModelWeightings) : void
RoomHardConstraints
#instance : RoomHardConstraints
-rhc_repo : IRepository<RoomHardConstraint>
+Instance() : RoomHardConstraints
+Kill() : void
+Insert(rhc : RoomHardConstraint) : void
+Delete(rhc : RoomHardConstraint) : void
+GetAll() : IEnumerable<RoomHardConstraint>
+GetById(id : int) : RoomHardConstraint
+EntryCount() : int
+HasRoomExclusivity(exam_id : int) : bool
+GetByType(type : RoomHardConstraint.types) : IEnumerable<RoomHardConstraint>
Solutions
#instance : Solutions
-solutions_repo : IRepository<Solution>
+Instance() : Solutions
+Kill() : void +Insert(solution : Solution) : void
+Delete(solution : Solution) : void
+GetAll() : IEnumerable<Solution>
+GetById(id : int) : Solution
PeriodHardConstraints
#instance : PeriodHardConstraints
-phc_repo : IRepository<PeriodHardConstraint>
+Instance() : PeriodHardConstraints
+Kill() : void +Insert(phc : PeriodHardConstraint) : void
+Delete(phc : PeriodHardConstraint) : void
+GetAll() : IEnumerable<PeriodHardConstraint>
+GetById(id : int) : PeriodHardConstraint
+GetByType(type : PeriodHardConstraint.types) : IEnumerable<PeriodHardConstraint>
+GetByTypeWithExamId(type : PeriodHardConstraint.types, exam_id : int) : IEnumerable<PeriodHardConstraint>
+GetExamsWithChainingCoincidence(exam_id : int) : IEnumerable<int>
-GetExamsWithChainingCoincidenceAux(exams : List<int>) : void
ConflictMatrix
#instance : ConflictMatrix
-conflict_matrix : int[,]
+Instance() : ConflictMatrix
+Kill() : void
+Get() : int[,]
+Set(conflict_matrix : int[,]) : void
Rooms
#instance : Rooms
-rooms_repo : IRepository<Room>
+Instance() : Rooms
+Kill() : void
+Insert(room : Room) : void
+Delete(room : Room) : void
+GetAll() : IEnumerable<Room>
+GetById(id : int) : Room
+EntryCount() : int

Figure 3.3: Business Layer

# 4

# Loader and solution initialization

The Loader and the solution initializer (heuristic) are the first tools used in this project. It is extremely necessary that the development of every tool and heuristic take in consideration the performance of its run. The less time a tool uses to execute, the more time other tools will use to do its job, and so may obtain better results. The Loader and Graph Coloring heuristic (solution initializer) will only be executed once so the major of the execution time will be used by the meta-heuristic(s).

## 4.1 Loader Module

The Loader module is the first tool to be used, above all. Its job is to load all the information presented in the set files. Each set file includes information about examinations and the students participating in each of it, the periods and the their penalties, the rooms and their penalties, period hard constraints, room hard constraints, and the information about the soft constraints, named Institutional Weightings. The presence of period and room hard constraints are optional.

This tool not only loads all the data to its corresponding repositories, but also creates and populates the conflict matrix depending on the data obtained previously. The conflict matrix is a matrix that has the information about the conflicts of each pair of examinations. This matrix is symmetric, for each pair of examinations has a conflict value regardless of its ordination, and so the conflict value of [i, j] is the same as [j, i]. This makes the population of the matrix twice as fast, because there's only needed to calculate the conflict once for each pair.

### 4.1.1 Analysis of benchmark data

It is very important to know each set by its difficulty. A set can be easy to find a feasible solution, but others no so much. Each set has different content, and so, some content may turn the set much harder than others. What turns some sets harder than others are the elevated number of students and examinations, which must be set in a rather limited number of rooms and time slots. A high conflict density is very problematic to create feasible solutions. Another aspect that makes sets harder is the number of hard constraints that much be followed. One example of hard set is the set 4, which has high conflict density and very limited number of rooms and time slots (in this case, only 1 room is available).

All the specifications and benchmark data from the 12 data sets of the ITC 2007 timetabling problem are shown in the Figure 4.1.

| | # students | # exams | # rooms | conflict matrix density | # time slots |
|---|---|---|---|---|---|
| Instance 1 | 7891 | 607 | 7 | 0.05 | 54 |
| Instance 2 | 12 743 | 870 | 49 | 0.01 | 40 |
| Instance 3 | 16 439 | 934 | 48 | 0.03 | 36 |
| Instance 4 | 5045 | 273 | 1 | 0.15 | 21 |
| Instance 5 | 9253 | 1018 | 3 | 0.009 | 42 |
| Instance 6 | 7909 | 242 | 8 | 0.06 | 16 |
| Instance 7 | 14 676 | 1096 | 15 | 0.02 | 80 |
| Instance 8 | 7718 | 598 | 8 | 0.05 | 80 |
| Instance 9 | 655 | 169 | 3 | 0.08 | 25 |
| Instance 10 | 1577 | 214 | 48 | 0.05 | 32 |
| Instance 11 | 16 439 | 934 | 40 | 0.03 | 26 |
| Instance 12 | 1653 | 78 | 50 | 0.18 | 12 |

Table 4.1: Specifications of the 12 data sets of the ITC 2007 examination timetabling problem.

### 4.1.2 Implementation

The development of the Loader tool is divided into two main parts: the implementation of the loading part, which loads the set file into the repositories using the business layer, and the creation and population of the conflict matrix.

First off, The Loader was implemented using the Loader base class, in which the LoaderTimetable extends. The Loader class implements functions to make it easier to run through a file. It uses StreamReader [**?**] to go through every line of the file and Regex [**?**] class to slit the phrases into tokens given a pattern. Loader offers the following methods: *NextLine*, *ReadNextToken*, *ReadCurrToken* and *ReadNextLine*. These methods are pretty useful to use directly in the LoaderTimetable class.

Unlike the Loader, LoaderTimetable depends on the structure of the set files. This class will use the Loader functions to go through a set file, and so, populate the repositories depending on the information read by the Loader class. LoaderTimetable offers the following operations: *Load*, *Unload*, *InitSolutions*, *InitInstitutionalWeightings*, *InitRoomHardConstraints*, *InitPeriodHardConstraints*, *InitRooms*, *InitPeriods* and *InitExaminations*, *InitConflictMatrix*. In this implementation only Load and Unload are public, while the rest is private, because their are only used within the class by the Load method.

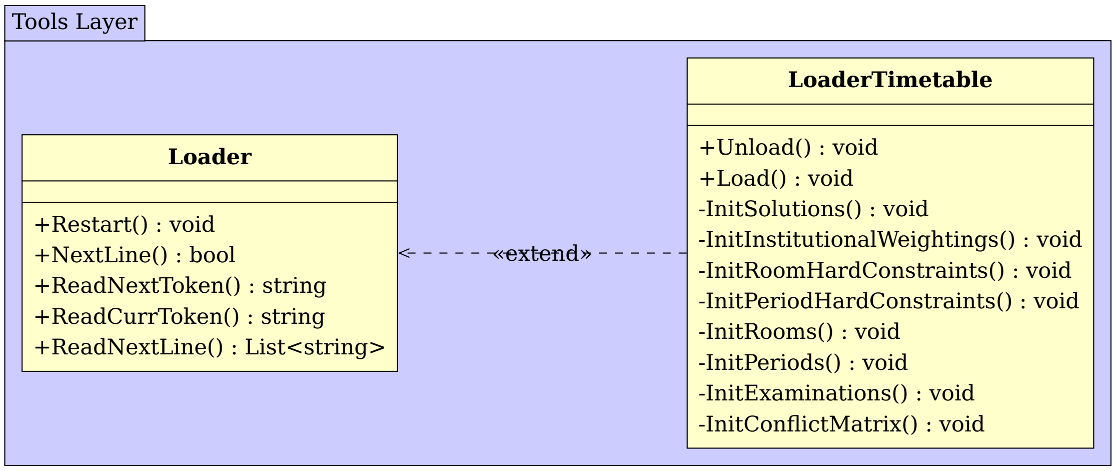
Tools Layer
Loader
+Restart() : void
+NextLine() : bool
+ReadNextToken() : string
+ReadCurrToken() : string
+ReadNextLine() : List<string>
«extend»
LoaderTimetable
+Unload() : void
+Load() : void
-InitSolutions() : void
-InitInstitutionalWeightings() : void
-InitRoomHardConstraints() : void
-InitPeriodHardConstraints() : void
-InitRooms() : void
-InitPeriods() : void
-InitExaminations() : void
-InitConflictMatrix() : void

Figure 4.1: Specification of Loader and LoaderTimetable tools

All specifications about these two classes can be seen in the Figure 4.1.

The implementation of LoaderTimetable class is all about the Load method. This public method will be "asking" for new lines and reading the tokens out of it, using the Loader class. This procedure will take place as long as there are new lines to read. It's a pretty simple cycle that gets a new line and checks if, for example, the string "Exams" is contained on that line. If so, it runs InitExaminations, if not, checks if "Periods" is contained on that line, and so on, until it runs out of file.

The pseudo code of this method can be seen on algorithm 1

**Algorithm 1** LoaderTimetabling's Loader method.

```
1: Read new line
2: repeat
3:     Read next token $token$
4:     If $token == null$ Then $break$
5:     If $token$ Contains "$Exams$" Then $InitExaminations()$
6:     Else If $token$ Contains "$Periods$" Then $InitPeriods()$
7:     Else If $token$ Contains "$Rooms$" Then $InitRooms()$
8:     Else If $token$ Contains "$PeriodHardConstraints$" Then $InitPeriodHardConstraints()$
9:     Else If $token$ Contains "$RoomHardConstraints$" Then $InitRoomHardConstraints()$
10:    Else If $token$ Contains "$InstitutionalWeightings$" Then $InitInstitutionalWeightings()$
11:    Else If Cannot read new line Then $break$
12: until Always
13: $InitSolutions()$
14: $InitConflictMatrix()$
```

## 4.2 Graph Coloring

- Largest degree ordering (LDO)

## 4.3 Experimental Evaluation/Results

**Algorithm 2** LoaderTimetabling's Loader method.

```
1: Read new line
2: repeat
3:     Read next token $token$
4:     If $token == null$ Then $break$
5:     If $token$ Contains "$Exams$" Then $InitExaminations()$
6:     Else If $token$ Contains "$Periods$" Then $InitPeriods()$
7:     Else If $token$ Contains "$Rooms$" Then $InitRooms()$
8:     Else If $token$ Contains "$PeriodHardConstraints$" Then $InitPeriodHardConstraints()$
9:     Else If $token$ Contains "$RoomHardConstraints$" Then $InitRoomHardConstraints()$
10:    Else If $token$ Contains "$InstitutionalWeightings$" Then $InitInstitutionalWeightings()$
11:    Else If Cannot read new line Then $break$
12: until Always
13: $InitSolutions()$
14: $InitConflictMatrix()$
```