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 Intercalar

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

Julho de 2015

Contents

List of Figures v

List of Tables vii

List of Code Listings ix

List of Acronyms ix

1 Introduction 1
- 1.1 Educational Timetabling Problems 1
- 1.2 Objectives 2
- 1.3 Planning 2
- 1.4 Document Organization 3

2 State-of-the-Art 5
- 2.1 Timetabling Problem 5
- 2.2 Existing Approaches 6
  - 2.2.1 Exact methods 7
    - Constraint-Programming Based Technique 7
    - Integer Programming 7
  - 2.2.2 Graph Coloring Based Techniques 8
    - Graph Coloring Problem 8
  - 2.2.3 Meta-heuristics 8
    - Single-solution meta-heuristics 8
    - Population-based meta-heuristics 8
  - 2.2.4 ITC2007 Examination timetabling problem: some approaches 9
  - 2.2.5 Other approaches 10

3 Architecture 15
- 3.1 System Architecture 15
  - 3.1.1 Data Layer 15
  - 3.1.2 Data Access Layer 15
  - 3.1.3 Business Layer 16
  - 3.1.4 Tools Layer 18
  - 3.1.5 Heuristics Layer 18

3.1.6 Presentation Layer 20

4 Loader and solution initialization 21
4.1 Loader Module 21
4.1.1 Analysis of benchmark data 21
4.1.2 Implementation 22
4.2 Graph Coloring 24
4.2.1 Implementation 24
4.3 Solution Initialization Results 25

5 Approach 1 - Local Search 29
5.1 Simulated Annealing 29
5.1.1 Implementation 30
5.2 Hill Climbing 33
5.3 Neighborhood Operators 33
5.3.1 Implementation 35

6 Experimental Results 37

7 Future Work 39

References 41

List of Figures

1.1 Gantt diagram of the project activities. 4

2.1 Optimization methods: taxonomy and organization (adapted from [1]). 6

3.1 Overview of subsystems that compose the system architecture. 16
3.2 Overview of the DAL and the present entity types and repositories. 17
3.3 Business Layer 19

4.1 Specification of Loader and LoaderTimetable tools 23
4.2 Graph Coloring and Feasibility Tester 26

5.1 simulated annealing classes 31
5.2 Simulated Annealing results 32
5.3 Neighborhood selection and operators 36

List of Tables

2.1 Timeline 14

4.1 Specifications of the 12 data sets of the ITC 2007 examination timetabling problem. 22
4.2 Graph Coloring's fitness and execution time 27

6.1 ITC 2007 results comparison to this project's approach. The best solutions are indicated in bold. "–" indicates that the corresponding instance is not tested or a feasible solution cannot be obtained. 38

List of Code Listings

1

Introduction

Many people believe that Artificial Intelligence (AI) 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. For hard problems like timetabling, job shop, traffic routing, clustering, among others, 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, in sports scheduling, or in 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.

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 or constraints. 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 type of rules/constraints to satisfy depend on the timetabling problem type and problem specifications. The constraints are generally divided into two constraint categories: hard and soft. 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 timetable generator tool. The problem at hand focus on the specifications introduced in the International Timetabling Competition 2007 (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 the 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 the progress report submitted in March, 2015. The second phase's main goal is to develop solutions for the ITC 2007 problem instance. The work developed in the second phase spanned from March to

July 2015, and is documented in the present document.

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. 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. Section 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 summarized. The next chapters address the implementation aspects of the solution method. This includes a chapter that explains the organization of the project and its architecture, following by chapters that explain the heuristics and the Loader module, ending with the experimental results and future work.

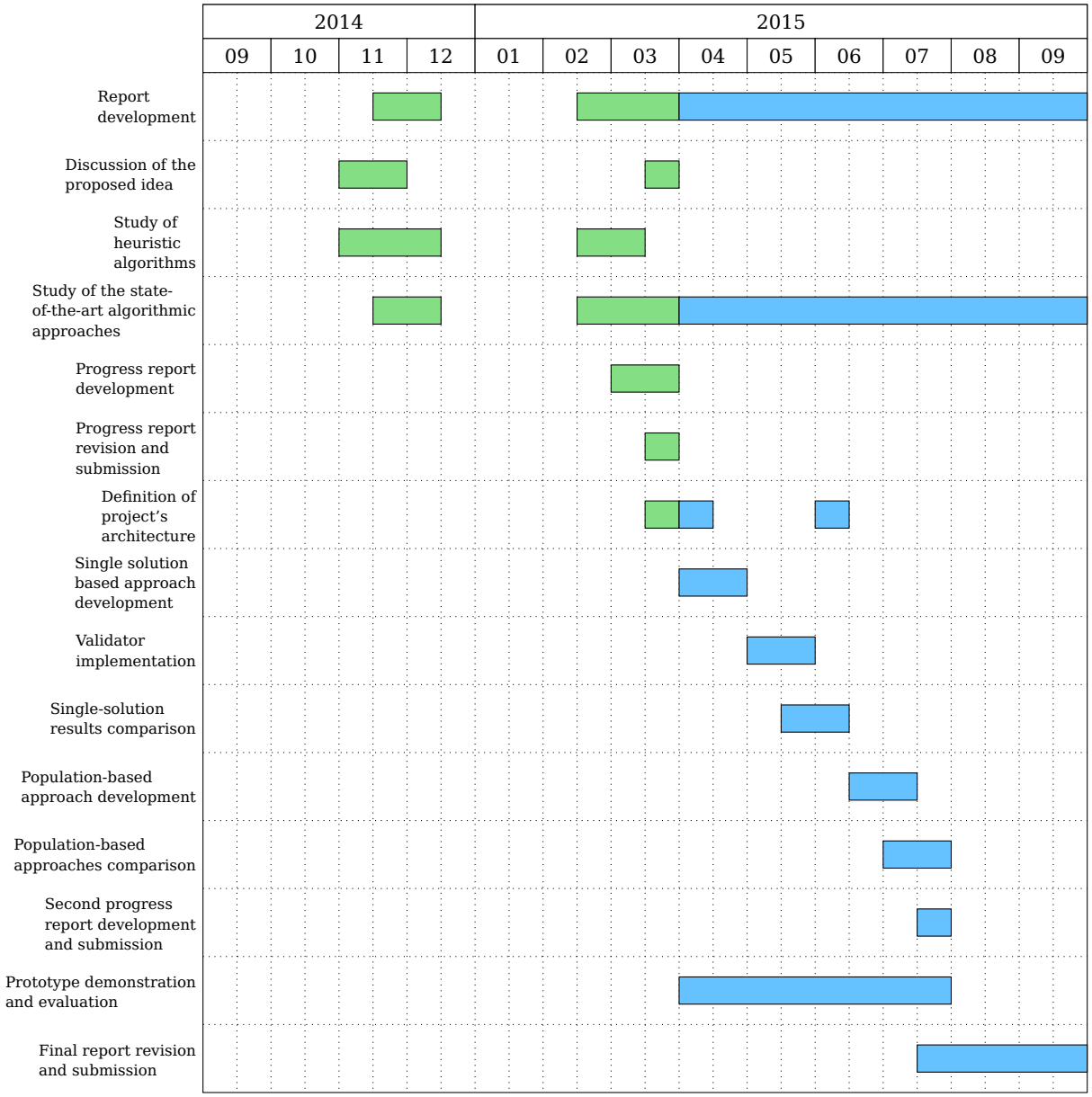

Figure 1.1: Gantt diagram of the project activities.

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 [2]. 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 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 [3], 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 [4] problem [5]. Considering the graph coloring as an optimization problem, it is proven that the task of finding the optimal solution is also a NP-Hard [4] problem [5]. Graph Coloring problems are explained further in Section 2.2

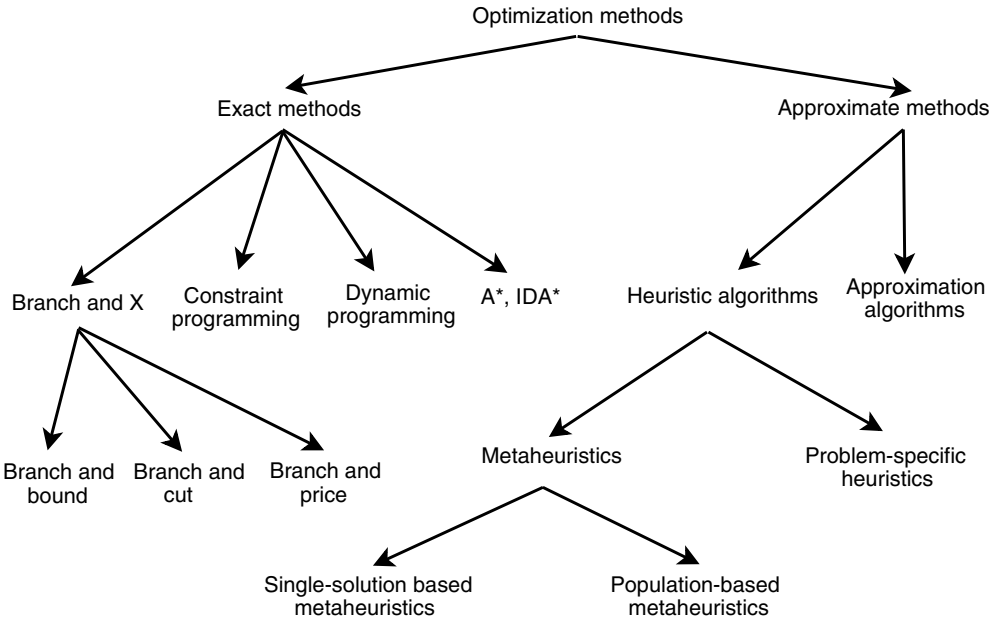

Figure 2.1: Optimization methods: taxonomy and organization (adapted from [1]).

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 [5]: *exact algorithms* (Branch-and-Bound, Constraint Programming), *graph based sequential techniques*, *Single-solution based meta-heuristics* (Tabu Search, Simulated Annealing, Great Deluge), *population based algorithms* (Evolutionary Algorithms, Memetic algorithms, Ant Colony algorithms, Artificial immune algorithms), *Multi-criteria techniques*, *Hyper-heuristics*, and *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 usually used to obtain a solution (feasible or not). For example, graph-coloring heuristics are used to obtain solutions for a given timetable problem instance. Usually only the hard constraints are considered in this phase. Meta-heuristics, on the other hand, are problem-independent, and are used to optimize any type problem. In these, one usually consider both

hard and soft requirements.

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 [6]. 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 satisfaction of the original constraints that were weakened, on a later stage of the algorithm.

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 an implicit 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, it may not be possible to use this approach 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 [5].

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 [2] 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 are 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 [5].

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 [7].

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* [1].

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* [1].

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* [1].

2.2.4 ITC2007 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 – the number of times a student is assigned to be in two exams that are N time slots apart should be minimized;
- mixed durations – the number of exams with different durations that occur in the same room and period should be minimized;
- 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 [8].

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 [9] 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 [10] to obtain feasible solutions and Conflict-based Statistics [11] to prevent IFS from looping. The second phase consists in using multiple optimization algorithms. These algorithms are applied in this order: Hill Climbing (HC) [12], Great Deluge (GD) [13], and optionally SA [14].

Gogos was able to reach second place in the Examination Timetabling track. Gogos' approach [15], like Müller's, is a two-phase approach. 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. The 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.

Atsuta et al. 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 [16] consists on applying a constraint satisfaction problem solver adopting an hybridization of TS and Iterated Local Search.

Geoffrey De Smet's approach [17] 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* [18]. 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 (TS) to improve the solutions obtained using the drools-solver.

Nelishia Pillay opted to use a two-phase algorithm variation, using a *Developmental Approach based on Cell Biology* [19], which 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

Abdullah et al.'s 2009 approach [20] 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 [21] to move

the solutions the region of optimal solutions.

McCollum et al.'s 2009 two phased approach [23] use an adaptive ordering heuristic from [24], proceeding with an *extended version* of GD. As the author stated, this approach is robust and general considering the obtained results on the benchmark datasets from ITC 2007.

Alzaqebah et al.'s 2011 two phased approach [25] 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.

Turabieh and Abdullah's 2011 approach [26] utilizes a 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). The second phase utilizes an hybridization of electromagnetic-like mechanism and GD algorithm, just like in [20].

Turabieh and Abdullah proposed another approach in 2012 [27] that utilizes a Tabu-based memetic algorithm which consists on an hybridization of a genetic algorithm with a Tabu Search algorithm. The authors state that this approach produces some of the best results when tested on ITC 2007's datasets.

Demeester et al. in 2012 created an hyper-heuristic approach [28]. The heuristics utilized are 'improved or equal' (equivalent to hill climbing that accepts equally good solutions), SA, GD and an adapted version of the *late acceptance* strategy [29]. 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.

McCollum et al.'s 2012 approach [30] introduce an IP formulation to the ITC 2007 instance, and also a solver using the CPLEX software.

Sabar et al.'s utilized a graphical coloring hyper-heuristic on its approach in 2012 [31]. 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.

Sabar et al.'s 2012 approach [32] 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, and largest enrollment first, applied in this order.

Abdullah and Alzaqebah opted to create an hybridization approach in 2013 [33], 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 [34] that utilizes an hybridization of a modified artificial bee colony with local search algorithm (i.e. late ac-

ceptance HC).

Burke et al.'s 2014 approach [35] 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 was 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*.

Hmer and Mouhoub's approach [36] 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.

Hamilton-Bryce et al. [37] in 2014 opted to use a non-stochastic method on their approach [37] 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. 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.

Rahman et al.'s approach [38] is a constructive one. 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.

Rahman et al.'s approach [39] utilizes adaptive linear combinations of graph coloring heuristics (largest degree and saturation degree) with an 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.

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 (creating the first cell, cell division, cell iteration and 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 SA.
	Abdullah et al. - Hybridization of electromagnetic-like mechanism and GD algorithm.
	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 three 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.

3

Architecture

In this chapter, a description of the architecture is given. The subsystems that compose the architecture are detailed. The proposed software architecture was designed taking into consideration aspects such as: code readability, extensibility, and efficiency.

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 Figure 3.1.

3.1.1 Data Layer

The DL is the layer where all entities are stored. The entities, which represent the different elements of a timetable, such as exam, room, or timeslot, are instantiated after reading an ITC 2007 benchmark file. Entities are maintained in memory and 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. This layer provides repositories of each type of entity. The repository was implemented in a way that its 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`.

The `Repository` class stores the entities in a list, with indexes corresponding to the entities identifiers. The `Repository` provides basic Create, Read, Update and Delete (CRUD) functions to access and edit the stored entities. The signature of the entities and all the specifications of the generic repository can be seen in the Figure 3.2.

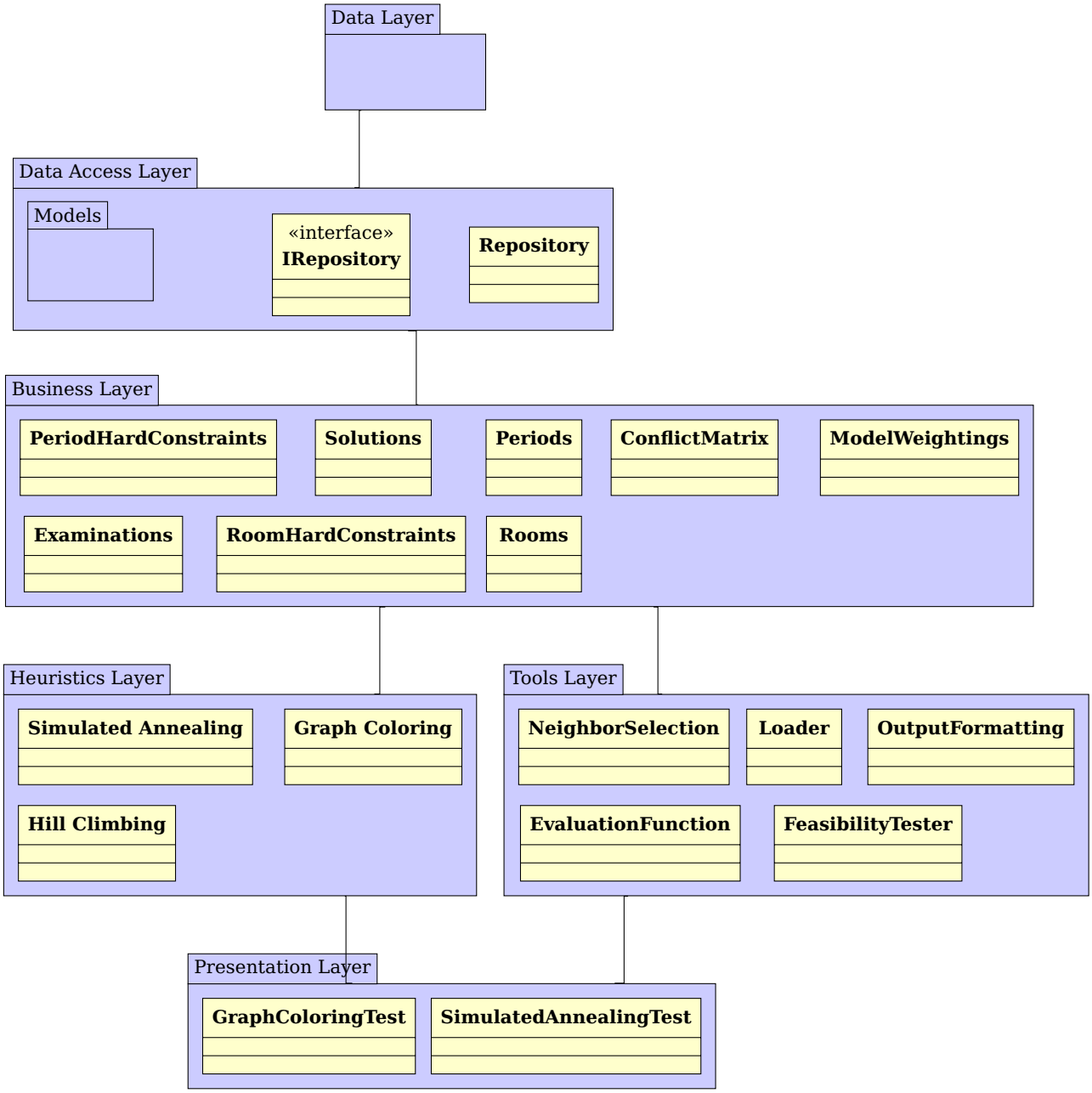

Figure 3.1: Overview of subsystems that compose the system architecture.

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 latter functions is the following. Considering the room hard constraints repository, one could invoke a method for obtaining all room

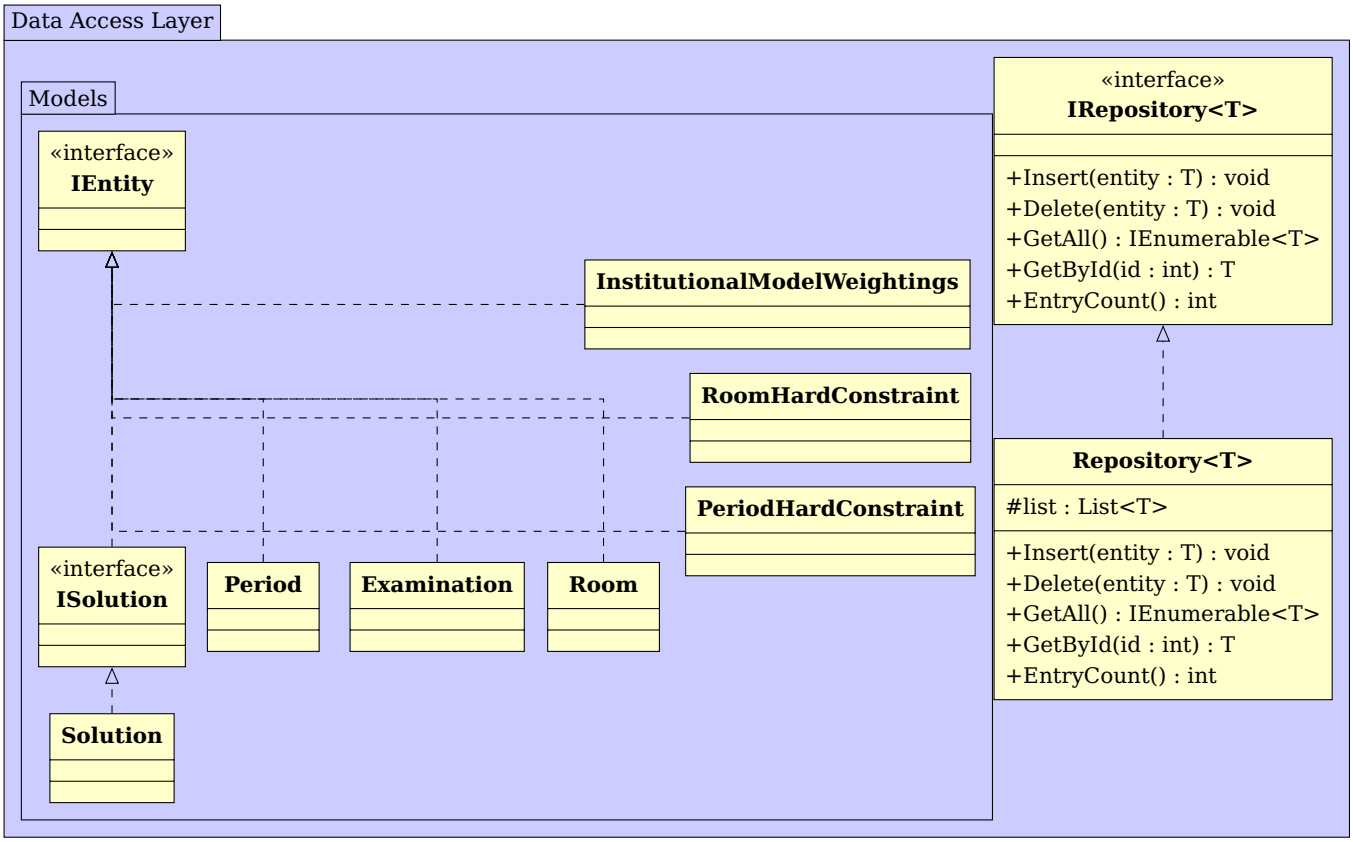

Figure 3.2: Overview of the DAL and the present entity types and repositories.

hard constraints of a given 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.

It's possible to store multiple instances of a certain type of entity, recurring to the BL classes, which provide `Repository` functions for that objective. If that's the case, a `Repository` instance of that type of entity is used. Thus, business classes that only store one instance, do not provide CRUD functions, just *set* and *get* functions. 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` class represents the *Conflict matrix*, of size equal to the number of examinations. Each matrix element (i, j) contains the number of conflicts between examinations i and j. 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 for solution validation, and computation of solution's fitness and distance to feasibility. A solution is only valid if the examinations are all scheduled, 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 metaheuristics used, such as SA, to compare different solutions.

The `Loader` loads all the information presented in a benchmark file 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 4.1.

The `NeighborSelection` is a tool that provides functions that verify 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 5.3.

The `FeasibilityTester` is a tool that provides functions, which efficiently checks if a certain examination can be placed in a certain period or room. An exam could be moved by only changing the period, the room, or both. This tool is used by both the GC and the 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 [41] in order to be able to submit the solution [42]. 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 they utilize the `NeighborSelection`

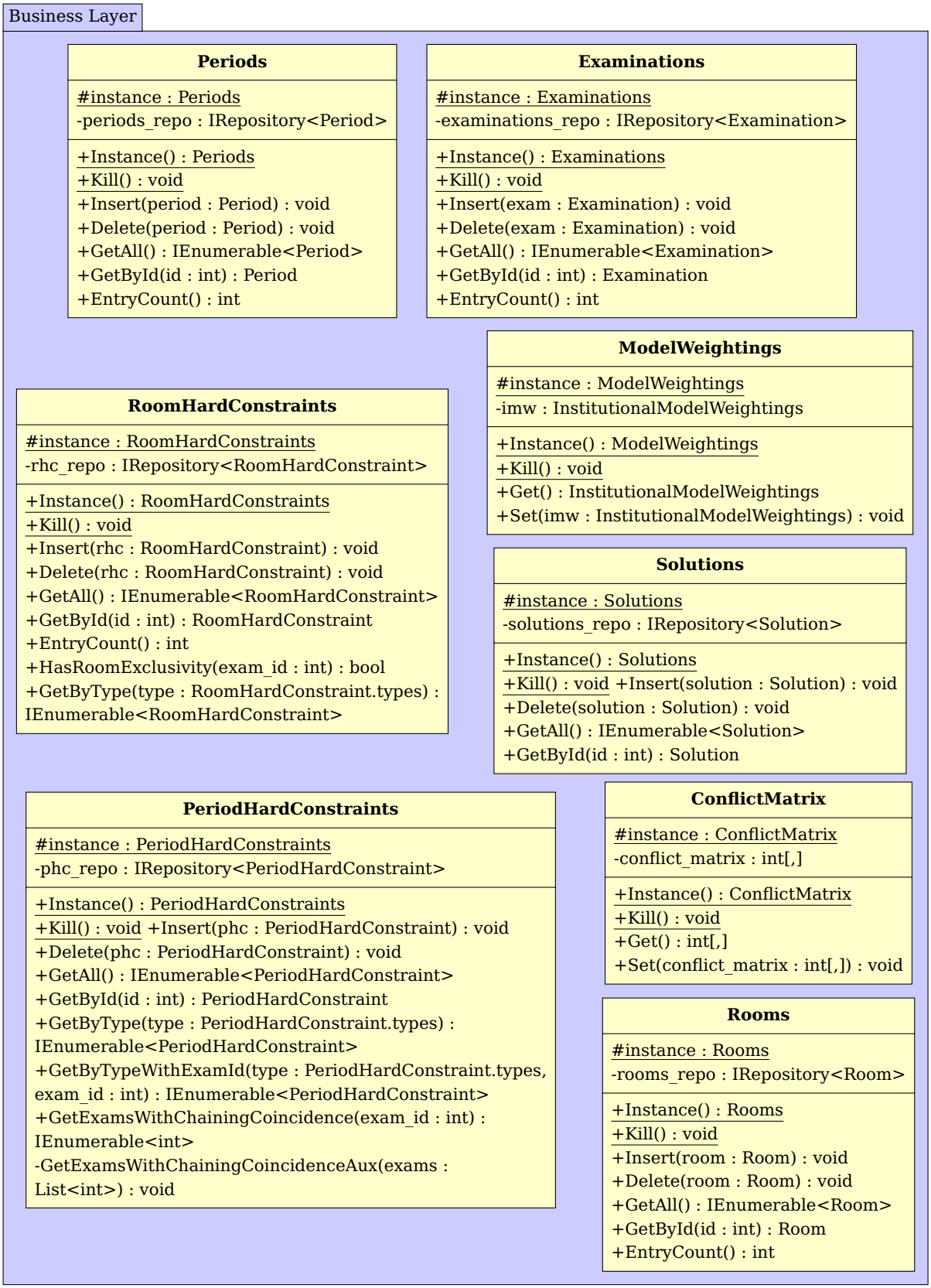

Figure 3.3: Business Layer

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 Chapters 4 and 5.

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, in a latter phase, another version of the presentation layer that includes the possibility of visualization of the final and best timetable generated.

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 [43] to go through every line of the file and Regex [44] 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.

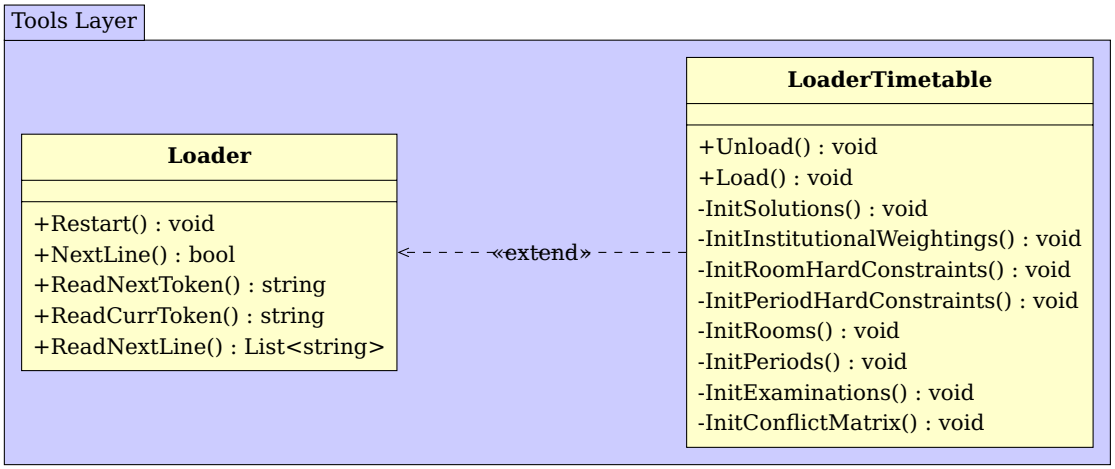

Figure 4.1: Specification of Loader and LoaderTimetable tools

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.

One very important topic on the LoaderTimetable class is the conflict matrix, as mentioned earlier. The conflict matrix is, as mentioned in Algorithm 1, created and populated in the Load method. The creation and population of the conflict matrix is what takes most of the

Algorithm 1 LoaderTimetabling's Loader method.

Read new line
repeat
  Read next token $token$
  If $token == null$ Then $break$
  If $token$ Contains "$Exams$" Then $InitExaminations()$
  Else If $token$ Contains "$Periods$" **Then** $InitPeriods()$
  Else If $token$ Contains "$Rooms$" **Then** $InitRooms()$
  Else If $token$ Contains "$PeriodHardConstraints$" **Then** $InitPeriodHardConstraints()$
  Else If $token$ Contains "$RoomHardConstraints$" **Then** $InitRoomHardConstraints()$
  Else If $token$ Contains "$InstitutionalWeightings$" **Then** $InitInstitutionalWeightings()$
  Else If Cannot read new line **Then** $break$
until Always
$InitSolutions()$
$InitConflictMatrix()$

time in the Load method, so it is crucial that it has the best performance possible.

In the first version of this method, InitConflictMatrix, the time of execution took beyond 2 seconds on most of the sets. The way the search was implemented was not the best, so it wasted too much time. The search was made in a way that for each pair of examinations, all the attendant students were tested to check for clashes.

The final implementation shows much better results. Instead of searching for all the students presented in each examination multiple times, this implementation organizes the students and examinations in a way that there's no need to search for the clashes. The method InitExaminations orders the examinations and its students as they are read. They are stored in a Hashmap, which stores the students as keys and the examinations which they attend as values. With this student-examinations organization, there's no need to search for the clashes, and so, the InitConflictMatrix simply goes to all pairs of examinations for each student, and add a conflict in the matrix for those pair of examinations. With this technique, the execution time of Load does not go beyond 80 milliseconds.

4.2 Graph Coloring

Graph Coloring is the heuristic used to generate a feasible solution. This heuristic is ran right after the loader. As the loader, it is convenient that this heuristic has the best possible performance, in order to give the next meta-heuristics most of the execution time. The implementation of this heuristic was based on Müller's approach [9].

4.2.1 Implementation

This heuristic is divided into phases. In the first phase, it starts by editing the conflict matrix, in order to add the exclusion hard constraints to the conflict matrix. This process is possible because the exclusion hard constraint is also a clash between a pair of examinations. This makes the algorithm easier to implement later on, because checking the conflict matrix for a clash between a pair of examinations now works for the student conflicts and exclusion.

The second phase simply erases all examination coincidence hard constraints' occurrences that has student conflicts. It is mentioned in the ITC 2007's site [45] that if two examinations have the examination coincidence hard constraint yet 'clash' with each other due to student enrollment, this hard constraint is ignored.

The third phase populates and sort the assignment lists. The assignment lists are four lists that has the unassigned examinations. These four lists contain:

- Unassigned examinations with "room exclusivity" hard constraint
- Unassigned examinations with "after" hard constraint
- Unassigned examinations with "examination coincidence" hard constraint
- All other unassigned examinations

The Largest Degree Ordering from Graph Coloring heuristic is used on these four lists, and so, each list is sorted by student conflict. This is very useful, because the examination assignment has list ordering as well. The order is mentioned above, and so, first all the examinations with room exclusivity are assigned, then all with after, and finally all with examination coincidence.

The fourth phase is the examination assignment phase. This phase is the most important phase of this heuristic. As mentioned it's based on Müller's approach [9]. It starts to assign the examinations with higher conflict, as mentioned above, using the four lists method. There are two types of assignment:

- Normal assignment - If it's possible to assign the chosen examination to a period and room, a normal assignment is processed. In this type of assignment, of all the possible periods to assign, one of them is chosen randomly. It should be noted that a possible period to assign means that the examination can be assigned to that period and at least in one room on that period. After choosing the period, the same will be done to the rooms, and so, a random room will be chosen from all possible assignable rooms for that examination and period. If the current examination has to be coincident to another, and so, set to the same period as the other, the rules explained won't be applied. Instead, only that period will be considered and if the period is not feasible to the current examination, the normal assignment won't occur.

- Forcing assignment - Occurs if there are no possible periods to assign the chosen examination, thus the normal assignment was not possible. A random period and room are selected and the examination will be forced to be assigned on those, unassigning all the examinations that conflict with this assignment. As the normal assignment, there are exceptions to this rule. If a coincident examination is already set, the examination to be set will be forced to be on the same period as the coincident examination, in a random room. The rule about force assigning an examination to specific period because of a coincident examination has 75% of chance to set the period the same as the coincident examination and 25% of chance to unassign all coincident examinations and try a random period instead. This *constraint* was applied to avoid infinite or very long finite loops, mainly in SET 6.

It is to be noted that the Graph Coloring heuristic implementation only signs, unassigns examinations and checks examination clashes when forcing an assignment. The feasibility checking done to periods and rooms (an examination being able to be set in a period and room) are made by the tool FeasibilityTester, as mentioned in the Section 3.1.3. The GraphColoring and FeasibilityTester classes, with all the variables and methods, can be seen in Figure 4.2. The algorithm of the graph coloring heuristic, which was explained in this section can be seen in Algorithm 2.

4.3 Solution Initialization Results

The results of the solution initialization are not meant to be the best, since the fitness (satisfying soft constraints) were not the objective of this first heuristic. Thus, it's rather impor-

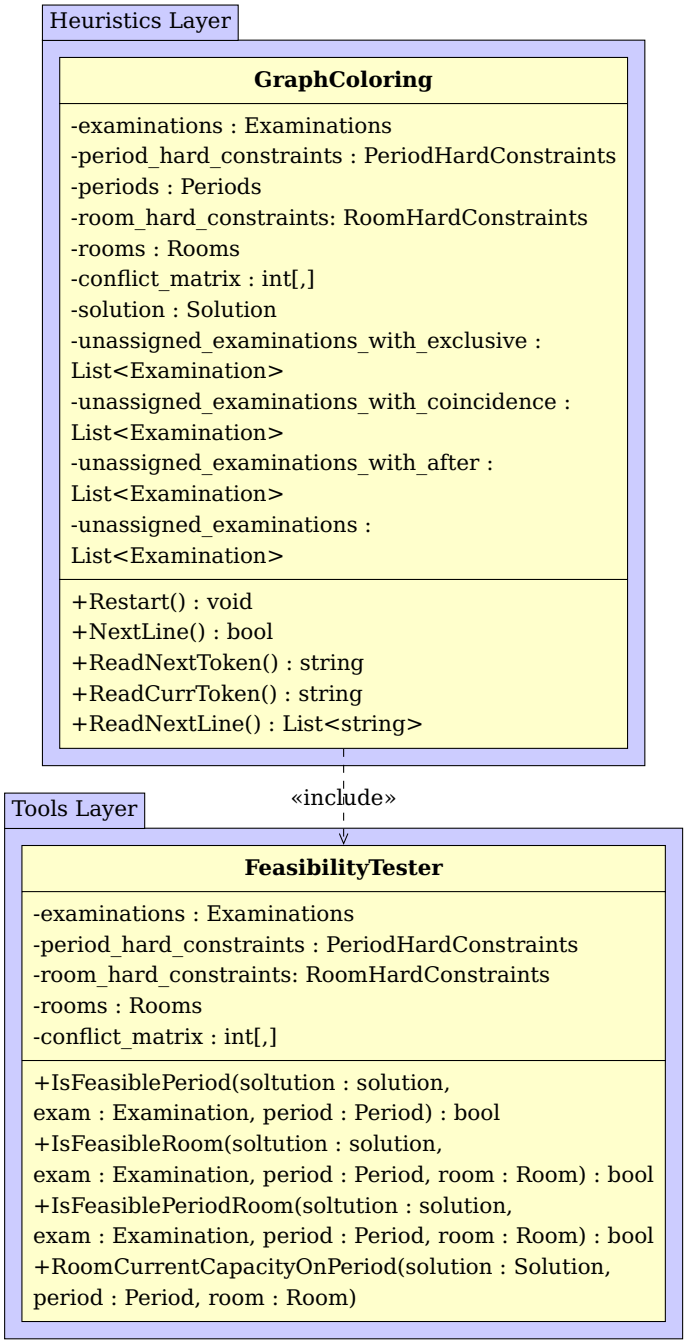

Figure 4.2: Graph Coloring and Feasibility Tester

Algorithm 2 Graph Coloring algorithm.

- Initial solution $solution$

1: Add exclusion to conflict matrix
2: Erase coincidence HC that contains conflict HC
3: Populate and sort assignment examination lists
4: **repeat**
5: Get the right list to use $list$
6: Remove last examination from $list$, $exam$
7: **If Not** NormalAssign($solution$, $exam$) **Then** ForceAssign($solution$, $exam$)
8: **until** No more examinations to assign
9: **Output:** $solution$

tant that the execution time and performance of this heuristic to be the best possible. Some sets are simpler than others, always getting good and stable execution times. Others can be harder, getting worse execution timings and instability. An unstable set means the results vary too much, and so some tests show very good execution times and others not so much. The Graph Coloring heuristic results on the 12 sets can be seen in Table 4.2. These results were obtained by running the heuristic 10 times for each set and compute its average for both fitness and execution time fields.

During this testing, we took the conclusion that the 6th set's results were unstable. Regardless of the results shown in the table 4.2, which only shows the average of 10 executions, some results in this set were 27, 30, 220, 213, 499 and 1212 milliseconds, which are not alike and prove instability. The same happened with the sets 11 and 12, which the set 11 was able to reach 9437 milliseconds in one of the tests, and the set 12 was able to reach 8001 milliseconds in one test as well.

As shown in the table 4.2, the heuristic could not get a feasible solution for the set 4. This

	Fitness	Execution time (ms)
SET 1	49 028	137
SET 2	103 907	250
SET 3	170 232	938
SET 4	–	–
SET 5	349 319	223
SET 6	60 857	298
SET 7	155 708	525
SET 8	397 868	232
SET 9	15 680	13
SET 10	121 164	110
SET 11	260 310	2481
SET 12	11 887	2413

Table 4.2: Graph Coloring's fitness and execution time

problem might be present because of the presence of an infinite cycle of normal and forcing assignments, resulting in assigning and unassigning the same examinations. This problem might be solved in the final version of this project.

5

Approach 1 - Local Search

The first approach consists about using a local search meta-heuristic(s) to improve the solution given by the graph coloring heuristic. This approach uses Simulated Annealing, based on Müller's approach [9], finishing with the use of Hill Climbing. Hill Climbing was added because the way Simulated Annealing is implemented, there's no way to have a good control of the execution time, and so the parameters are given to make it run almost all the limit time and the rest is executed on Hill Climbing which the execution time is perfectly controllable.

5.1 Simulated Annealing

Simulated Annealing is a single-solution meta-heuristic (section 2.2.3). This meta-heuristic optimizes a solution by generating neighbor solutions which might be accepted given a acceptance criteria. A neighbor solution is the application of a neighbor operator to the current solution, creating a new solution which is a neighbor of the current solution. A neighbor operator, in this context, could be the movement of an examination to another time slot. Being a single-solution meta-heuristic, it only generates one neighbor at the time. The neighbor operators and acceptance criteria are the most important part of this algorithm. Little changes on one of these may get the algorithm behave in very different ways and end up with much different solutions.

The acceptance criteria will, considering the current solution and a neighbor solution, give the percentage of acceptance of the neighbor solution. Most of the approaches using this meta-heuristic accept a new neighbor solution if this is *better* than the current solution. Otherwise, there's a chance that the neighbor solution is still accepted, depending on certain parameters. The parameters of the acceptance criteria are the *Temperature* (normally given as maximum and minimum temperature) and the *Cooling Schedule*. By definition, the higher the temperature, the higher is the chance to accept a worse solution over the current solution. The cooling schedule, as the name suggests, is a function that lowers the temperature. The SA algorithm finishes when the current temperature is lower or equal to the minimum temperature. The temperature should start high enough to accept all possible wrong solutions at the beginning, in order to search the maximum value of solutions in the solution space.

5.1.1 Implementation

The simulated annealing was implemented in a way that it is independent from the type of the cooling schedule and neighbor generator. The simulated annealing base class is abstract and implements everything except for the neighbor generator, which is an abstract method that must be implemented in order to decide how the neighbor generator behaves. It also does not implement the evaluation function (the one that computes the fitness value of a solution or neighbor). The SimulatedAnnealing and SimulatedAnnealingTimetable's methods and variables can be seen in the figure 5.1.

The SimulatedAnnealing abstract class has the methods *Exec*, *Exec2*, *OnlyBetter* and *ExecLinearTimer*, which are all similar, but were created to test different approaches. All these methods share the same code, in exception to the cooling schedule (the way the temperature is updated) and acceptance criteria. The pseudo code of these can be seen in Algorithm 3.

The ExecLinearTimer has a linear cooling schedule, which is directly proportional to the spent time, and uses the following acceptance criteria:

$$P(\delta E, T) = e^{\frac{-\delta E}{T}}$$

T $\rightarrow$ Current temperature
δE $\rightarrow$ Fitness difference between the new neighbor and current solution

The Exec method shares the same acceptance criteria but uses geometric cooling schedule:

$$T = T.r$$

Algorithm 3 Simulated Annealing method.

Input:

- s // Initial solution
- $TMax$ // Maximum temperature
- $TMin$ // Minimum temperature
- $loops$ // Number of loops per temperature

$T = Tmax$; // Starting temperature
Ac ; // Acceptance criteria initializer
repeat
  repeat
    Generate a random neighbor s';
    $\delta E = f(s') - f(s)$;
    If $E \leq 0$ **Then** $s = s'$ // Accept the neighbor solution
    Else Accept s' with a probability computed using the Ac;
  until Number of iterations reached $loops$
  $T = g(T)$ // Temperature update
until $T < TMin$
Output: s // return the current (best) solution

r → rate

The rate must belong in the interval]0,1[. The closer to 1, the longer the algorithm takes to finish and wider is the area of solutions to be analyzed in the solution space.

The Exec2 method is the one used in this project. It uses an exponential (decreasing) cooling schedule [46]:

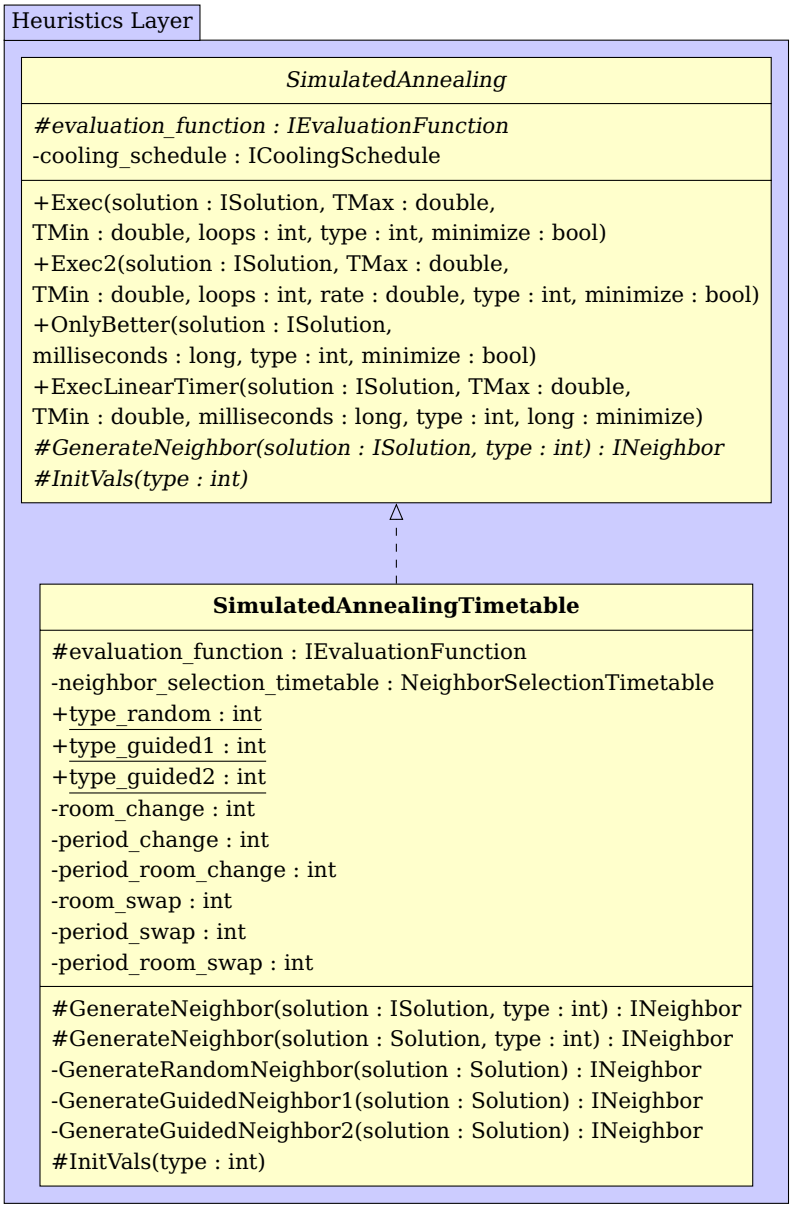

Figure 5.1: simulated annealing classes

$$T = T_{max} e^{-R.t}$$

t $\rightarrow$ Current span, counter stated from 0
T_{max} $\rightarrow$ Maximum/initial temperature
R $\rightarrow$ Decreasing rate

This method also uses a different acceptance criteria:

$$P(\delta E, T, SF) = e^{\frac{-\delta E}{T \cdot f(s)}}$$

f(s) $\rightarrow$ Solution fitness

Some testings on the parameters were performed, so to check what parameters gave the best results. All the testings were made for the first set only (these testings require a lot of time), and so the same parameters will be used on all the sets so we can compare to results to other approaches, mainly the first winners of the competition. In the Figure 5.2 is an example of a plot of the SA running with the following parameters: *TMax* = 45, *TMin* = e^{-18}, *loops* = 5 and *rate* = 0.01.

As can be seen in the Figure 5.2, it starts by accepting almost all better neighbor solutions, because in the beginning they are rather easy to find and are all accepted. In the middle, the score of the accepted neighbors are stable, because it accepts almost all worst solutions and the ratio of seeking better or worse solutions is very similar. In the end, the temperature is so low it becomes harder to accept worse solutions, ending up acting similar to a Hill Climbing.

The tests performed on all the sets are shown and explained in the section 6

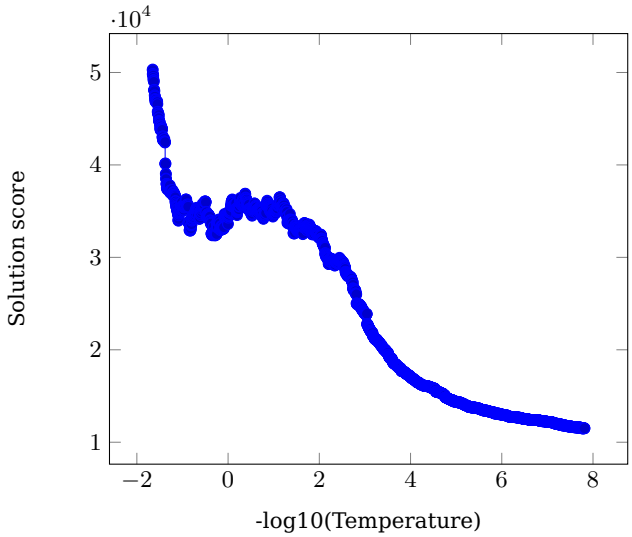

Figure 5.2: Simulated Annealing results

5.2 Hill Climbing

Hill Climbing is a similar meta-heuristic compared to simulated annealing. The only difference is that it doesn't need acceptance criteria. Not needing this, means it doesn't actually need temperature and cooling schedule to be executed. Not being stuck with these parameters, it can be parameterized directly to be executed in a time limit. After the neighbor creation and acceptance/rejection the elapsed time is checked, and so the method keeps running or not depending if the elapsed time already passed the limit time (normally the time limit given is little shorter than the original limit time).

Considering the main meta-heuristic used being the simulated annealing, this basically works like the hill climbing in the end, because of the presence of the low temperature. So, as the execution time could not be directly manipulated in the simulated annealing, the hill climbing was added to continue the execution until the time limit expires, since it can be controlled in this last meta-heuristic.

The hill climbing method is present in the simulated annealing abstract class, which the name of the method is *OnlyBetter*. Works the same way as the other normal simulated annealing methods, sharing the neighbor operators, but in the input parameters is present the limit execution time, and as said above, does not accept worse solutions.

5.3 Neighborhood Operators

Neighborhood operators are operations made to a solution, in order to create other valid solutions (neighbor solutions), but not necessarily feasible. In this context, the core of all operations are the relocation of the examinations.

The implementation of neighborhood selection went through tree different approaches. Firstly, the random selection was implemented. This approach always chooses a random operator to generate a new neighbor. Thereafter two guided approaches were implemented. The first one raised the probability of selecting one operator if this one generated a better neighbor solution. The probability of that operator is reduced in an equal amount if the operator generated a worse solution. The other guided approach simply lower the probability of an operator in case of success, and raises in case of unsuccess.

The second guided approach presented worse results compared to the first one, as expected. Oddly, the random approach almost always showed better results compared to the first guided approach, even after suffering major changes in order to try to get it to generate better results. These changes include raising the probabilities even more or less when successfully or unsuccessfully created a better solution, and instead of lowering the probability for an operator when created a worse solution, it was instead returned to its default value.

The neighborhood operators, in this context, are all based on moving examinations to another *spot*. This implementation uses six different neighborhood operators:

- *Room Change* - A random examination is selected. After that, a random room is randomly selected. If the assignment of the random examination to the random room, while maintaining the period, does not clash any hard constraints, that neighbor is returned. If not, the next rooms are tested until one of them creates a feasible solution. If it reaches the limit of rooms and no feasible solution was found, no neighbor is returned;

- *Period Change* - A random examination is selected. After that, a random period is randomly selected. If the assignment of the random examination to the random period, while maintaining the room, does not clash any hard constraints, that neighbor is returned. If not, the next periods are tested until one of them creates a feasible solution. If it reaches the limit of periods and no feasible solution was found, no neighbor is returned;

- *Period & Room Change* - A random examination is selected. After that, a random room and period are randomly selected. If the assignment of the random examination to the random room and period does not clash any hard constraints, that neighbor is returned. If not, the next rooms are tested for each of the next periods, until one of them creates a feasible solution. If it reaches the limit of periods and rooms and no feasible solution was found, no neighbor is returned;

- *Room Swap* - A random examination is selected. After that a random room is selected. If the selected examination can be placed in that room, while maintaining the period, then a *Room Change* neighbor is returned instead. If not, if the swapping of the random examination with any of the examinations presented in the random room, keeping the same period, does not clash any hard constraints, that neighbor is returned. If not, the examinations presented in the next rooms are tested until a feasible solution is found (always testing first if a *Room Change* can be returned instead). If it reaches the limit of rooms and no feasible solution was found, no neighbor is returned;

- *Period Swap* - A random examination is selected. After that a random period is selected. If the selected examination can be placed in that period, while maintaining the room, then a *Period Change* neighbor is returned instead. If not, if the swapping of the random examination with any of the examinations presented in the random period, keeping the same room, does not clash any hard constraints, that neighbor is returned. If not, the examinations presented in the next periods are tested until a feasible solution is found (always testing first if a *Period Change* can be returned instead). If it reaches the limit of periods and no feasible solution was found, no neighbor is returned;

- *Period & Room Swap* - A random examination is selected. After that a random period and room are selected. If the selected examination can be placed in that period and room, then a *Period Change* neighbor is returned instead. If not, if the swapping of the random examination with any of the examinations presented in the random period and room does not clash any hard constraints, that neighbor is returned. If not, the examinations presented in the next periods and rooms are tested until a feasible solution is found (always testing first if a *Period & Room Change* can be returned instead). If it reaches the limit of periods and rooms and no feasible solution was found, no neighbor is returned;

5.3.1 Implementation

The original concept of neighbor solution is to have another solution apart from the current one, which is the result of applying the neighborhood operator to the current solution. But in this project, a neighbor solution is not really a solution, but the changes that need to be done to the original solution in order to obtain the neighbor solution itself. This makes the "job" easier and faster, since there's no need to replicate the original solution and apply the neighborhood operator to it in order to obtain the neighbor solution. Instead, the neighbor objects presented in the project have the information about the original solution and the operations needed to be done in order to change the original solution (what is normally called as "replacing the original solution with the neighbor"). In this case, there's no replacing the original solution with the neighbor, but *accepting* the neighbor solution will just apply the operator to the current solution.

Every neighbor object must implement the interface *INeighbor*, which presents the methods *Accept*, *Reverse* and an integer that represents the fitness of the neighbor (the fitness of the new solution if this neighbor is to be accepted). The method Accept applies the operation to the current solution, modifying it and act like it's replacing the solution with the neighbor solution. The Reverse method disapplies the operation, getting then the old solution. The different neighbors and its methods can be seen in the Figure 5.3.

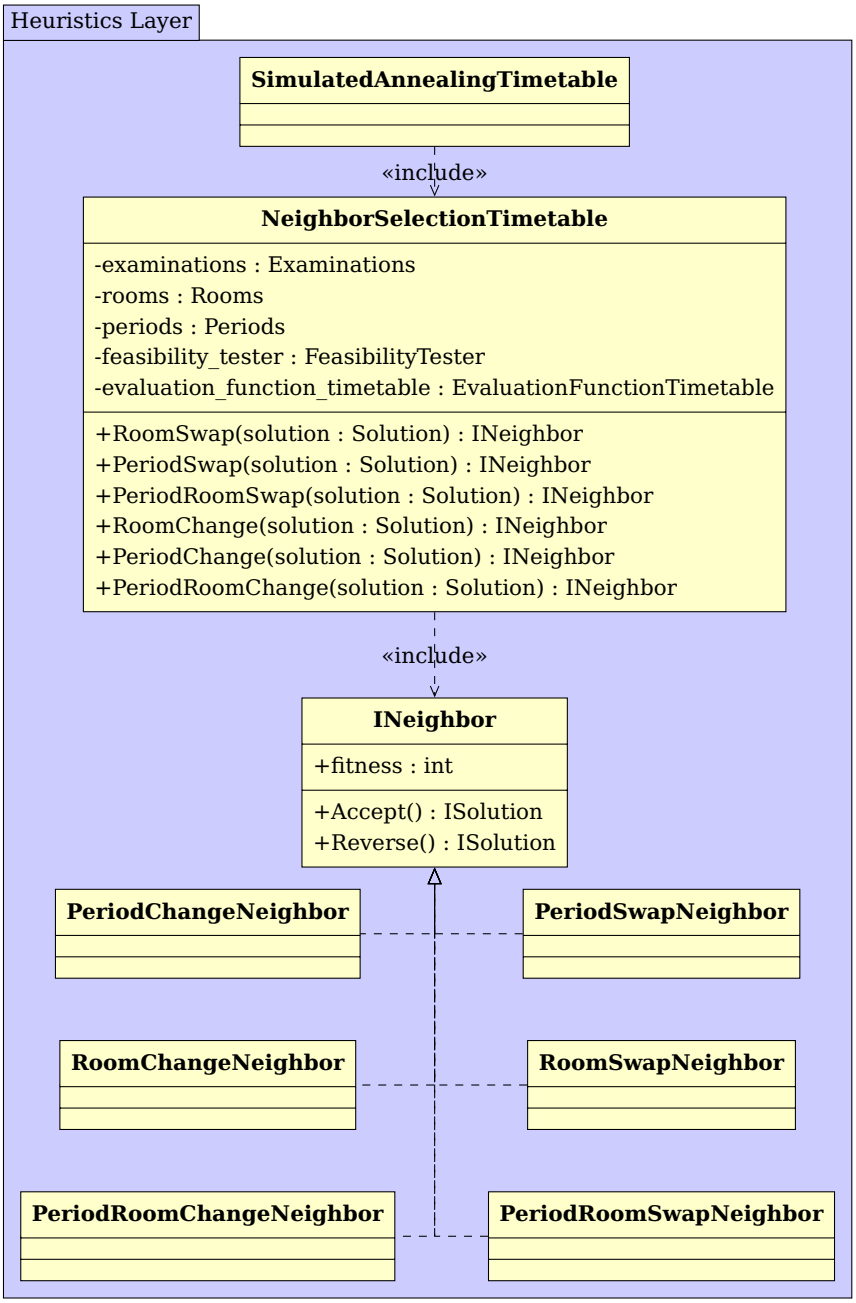

Figure 5.3: Neighborhood selection and operators

6

Experimental Results

In this section we will present the results obtained while running all the heuristics explained in the previous chapters. We will also compare the obtained results with the top 5 winners of the ITC2007.

The results obtained using the approach explained in the previous chapters were not as we expected. Some of the results obtained could be as good as it can match the top 5 winners, and some were so far away from these. Each set score is the result of the average of 10 runs that were done on that same set. The tests were made on all 12 sets, including the hidden ones.

The parameters used to run the Simulated Annealing and the Hill Climbing were:

TMax $\rightarrow$ 0.01
TMin $\rightarrow 1e^{-18}$
reps $\rightarrow$ 5
rate $\rightarrow$ 0.001

With these parameters used on all the sets, we obtained the results presented on the Table 6.1. As can be seen, the results for the sets 1, 2, 6, 7, 8 and 9 can match the 5 winners' results, but the sets 3, 10 and 11 were way too far away from the results of the top 5 (the ones that could get a feasible solution). We believe this can be fixed by tweaking the parameters and give individual parameters to each set to try and obtain better results.

Instance	Müller 2008 [9]	Gogos et al. 2008 [15]	Atsuta et al. 2008 [16]	De Smet 2008 [17]	Pillay 2008 [19]	My Approach 2015
1	**4370**	5905	8006	6670	12 035	6934
2	**400**	1008	3470	623	3074	821
3	**10 049**	13 862	18 622	–	15 917	24 627
4	**18 141**	18 674	22 559	–	23 582	–
5	**2988**	4139	4714	3847	6860	7729
6	**26 950**	27 640	29 155	27 815	32 250	30 195
7	**4213**	6683	10 473	5420	17 666	8089
8	**7861**	10 521	14 317	–	16 184	11 067
9	**1047**	1159	1737	1288	2055	1448
10	16 682	–	15 085	**14 778**	17 724	35 698
11	**34 129**	43 888	–	–	40 535	72 751
12	5535	–	**5264**	–	6310	8049

Table 6.1: ITC 2007 results comparison to this project's approach. The best solutions are indicated in bold. "–" indicates that the corresponding instance is not tested or a feasible solution cannot be obtained.

7

Future Work

In the future work, we plan on getting better results on all the sets by tweaking some features in the implemented code. This includes changing the input parameters for each set to obtain the best performance in order to deliver the best results on each one of them.

One of the main features that needs to be upgraded is the performance of the simulated annealing (and of course the hill climbing which in this case is part of the simulated annealing), because this heuristic for each new generated neighbor, it computes its fitness value again, from the beginning. As some tests were made, we concluded that most of the time spent on this heuristic is used by the computation of the fitness value for each neighbor. We should get better results if the fitness value was computed based on the change that was made on the new neighbor, instead of computing it all again. First we need to study if this feature is possible to implement. If it is, the performance boost should be very noticeable.

The graph coloring heuristic will probably be edit in order to try and get a feasible solution on all the sets, since the 4th set probably ends up in an infinite loop, and so it gets unfeasible.

Apart from the changes on the existing code, if there's time to implement new heuristics to test and compare to the top 5 winners, we plan on implementing a population-based algorithm like the Genetic Algorithm or a to be used as optimization methods after the already implemented graph coloring.

References

1. E. Talbi, *Metaheuristics - From Design to Implementation*. Wiley, 2009.
2. A. Schaerf, "A survey of automated timetabling," *Artif. Intell. Rev.*, vol. 13, no. 2, pp. 87–127, 1999.
3. T. R. Jensen, *Graph Coloring Problems*. John Wiley & Sons, Inc., 2001.
4. S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
5. R. Qu, E. Burke, B. McCollum, L. Merlot, and S. Lee, "A survey of search methodologies and automated system development for examination timetabling," *J. Scheduling*, vol. 12, no. 1, pp. 55–89, 2009.
6. R. Lewis, "A survey of metaheuristic-based techniques for university timetabling problems," *OR Spectrum Volume 30, Issue 1 , pp 167-190*, 2007.
7. M. Carter, G. Laporte, and S. Lee, "Examination timetabling: Algorithmic strategies and applications," *The Journal of the Operational Research Society*, 1996.
8. B. McCollum, "Itc2007 examination evaluation function." `http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/examevaluation.htm`, 2007.
9. T. Müller, "ITC2007 solver description: A hybrid approach," *Annals of Operations Research*, vol. 172, no. 1, pp. 429–446, 2009.
10. T. Müller, *Constraint-Based Timetabling*. PhD thesis, Charles University in Prague Faculty of Mathematics and Physics, 2005.
11. T. Müller, R. Barták, and H. Rudová, *Conflict-Based Statistics*. PhD thesis, Faculty of Mathematics and Physics, Charles University Malostranské nám. 2/25, Prague, Czech Republic, 2004.
12. S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.
13. G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *Journal of Computational Physics*, 1993.
14. S. Kirkpatrick, D. Gelatt, and M. Vecchi, "Optimization by simmulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
15. C. Gogos, P. Alefragis, and E. Housos, "An improved multi-staged algorithmic process for the solution of the examination timetabling problem," *Annals of Operations Research*, vol. 194, no. 1, pp. 203–221, 2012.
16. M. Atsuta, K. Nonobe, and T. Ibaraki, *ITC-2007 Track2: An Approach using General CSP Solver*. PhD thesis, Kwansei-Gakuin University, School of Science and Technology, Tokyo, Japan, 2007.
17. G. Smet, "Drools-solver." `http://www.cs.qub.ac.uk/itc2007/winner/bestexamsolutions/Geoffrey_De_smet_examination_description.pdf`, 2007.
18. T. J. Drools, "Drools planner user guide." `http://docs.jboss.org/drools/release/5.4.0.Final/drools-planner-docs/html_single/`.
19. N. Pillay, "A developmental approach to the examination timetabling problem." `http://www.cs.qub.ac.uk/itc2007/winner/bestexamsolutions/pillay.pdf`, 2007.

20. S. Abdullah, H. Turabieh, and B. McCollum, "A hybridization of electromagnetic-like mechanism and great deluge for examination timetabling problems," in *Hybrid Metaheuristics, 6th International Workshop, HM 2009, Udine, Italy, October 16-17, 2009. Proceedings* (M. J. Blesa, C. Blum, L. D. Gaspero, A. Roli, M. Sampels, and A. Schaerf, eds.), vol. 5818 of *Lecture Notes in Computer Science*, pp. 60–72, Springer, 2009.
21. N. Javadian, M. Alikhani, and R. Tavakkoli-Moghaddam, "A discrete binary version of the electromagnetism-like heuristic for solving traveling salesman problem," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence, 4th International Conference on Intelligent Computing, ICIC 2008, Shanghai, China, September 15-18, 2008, Proceedings* (D. Huang, D. C. W. II, D. S. Levine, and K. Jo, eds.), vol. 5227 of *Lecture Notes in Computer Science*, pp. 123–130, Springer, 2008.
22. C. Gogos, G. Goulas, P. Alefragis, and E. Housos, "Pursuit of better results for the examination timetabling problem using grid resources," in *2009 IEEE Symposium on Computational Intelligence in Scheduling, CISched 2009, Nashville, TN, USA, April 1-2, 2009*, pp. 48–53, IEEE, 2009.
23. B. McCollum, P. McMullan, A. Parkes, E. Burke, and S. Abdullah, "An extended great deluge approach to the examination timetabling problem," in *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), 10-12 Aug 2009, Dublin, Ireland* (J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, eds.), pp. 424–434, 2009. Paper.
24. E. Burke and J. Newall, "Solving examination timetabling problems through adaption of heuristic orderings," *Annals of Operations Research*, vol. 129, no. 1-4, pp. 107–134, 2004.
25. M. Alzaqebah and S. Abdullah, "Hybrid artificial bee colony search algorithm based on disruptive selection for examination timetabling problems," in *Combinatorial Optimization and Applications - 5th International Conference, COCOA 2011, Zhangjiajie, China, August 4-6, 2011. Proceedings* (W. Wang, X. Zhu, and D. Du, eds.), vol. 6831 of *Lecture Notes in Computer Science*, pp. 31–45, Springer, 2011.
26. H. Turabieh and S. Abdullah, "An integrated hybrid approach to the examination timetabling problem," *Omega*, vol. 39, no. 6, pp. 598 – 607, 2011.
27. S. Abdullah and H. Turabieh, "On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems," *Information Sciences*, vol. 191, no. 0, pp. 146 – 168, 2012. Data Mining for Software Trustworthiness.
28. P. Demeester, B. Bilgin, P. Causmaecker, and G. Berghe, "A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice," *J. Scheduling*, vol. 15, no. 1, pp. 83–103, 2012.
29. E. Burke and Y. Bykov, "A late acceptance strategy in hill-climbing for examination timetabling problems," in *In Proceedings of the conference on the Practice and Theory of Automated Timetabling(PATAT*, 2008.
30. B. McCollum, P. McMullan, A. Parkes, E. Burke, and R. Qu, "A new model for automated examination timetabling," *Annals of Operations Research*, vol. 194, no. 1, pp. 291–315, 2012.
31. N. Sabar, M. Ayob, R. Qu, and G. Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Appl. Intell.*, vol. 37, no. 1, pp. 1–11, 2012.
32. N. Sabar, M. Ayob, G. Kendall, and R. Qu, "A honey-bee mating optimization algorithm for educational timetabling problems," *European Journal of Operational Research*, vol. 216, no. 3, pp. 533–543, 2012.
33. S. Abdullah and M. Alzaqebah, "A hybrid self-ddaptive bees algorithm for examination timetabling problems," *Appl. Soft Comput.*, vol. 13, no. 8, pp. 3608–3620, 2013.
34. M. Alzaqebah and S. Abdullah, "An adaptive artificial bee colony and late-acceptance hill-climbing algorithm for examination timetabling," *J. Scheduling*, vol. 17, no. 3, pp. 249–262, 2014.
35. E. Burke, R. Qu, and A. Soghier, "Adaptive selection of heuristics for improving exam timetables," *Annals of Operations Research*, vol. 218, no. 1, pp. 129–145, 2014.

36. A. Hmer and M. Mouhoub, "A multi-phase hybrid metaheuristics approach for the exam timetabling," *Department of Computer Science University of Regina Regina, Canada fhmer200a,mouhoubmg@cs.uregina.ca*, 2014.
37. R. Hamilton-Bryce, P. McMullan, and B. McCollum, "Directed selection using reinforcement learning for the examination timetabling problem," 2014.
38. S. Rahman, E. Burke, A. Bargiela, B. McCollum, and E. Özcan, "A constructive approach to examination timetabling based on adaptive decomposition and ordering," *Annals of Operations Research*, vol. 218, no. 1, pp. 3–21, 2014.
39. S. Rahman, A. Bargiela, E. Burke, E. Özcan, B. McCollum, and P. McMullan, "Adaptive linear combination of heuristic orderings in constructing examination timetables," *European Journal of Operational Research*, vol. 232, no. 2, pp. 287–297, 2014.
40. B. McCollum, "Itc2007 examination timetabling datasets." `http://www.cs.qub.ac.uk/itc2007/Login/SecretPage.php`, 2007.
41. B. McCollum, "Itc2007 examination output format." `http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/outputformat.htm`, 2007.
42. B. McCollum, "Itc2007 examination validator." `http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/validation.htm`, 2007.
43. Microsoft, "Streamreader class." `https://msdn.microsoft.com/en-us/library/system.io.streamreader(v=vs.110).aspx`, 2015.
44. Microsoft, "Regex class." `https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex(v=vs.110).aspx`, 2015.
45. B. McCollum, "Itc2007 examination input format." `http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/Inputformat.htm`, 2007.
46. P. Carvalho, *Lecture Notes in Evolutionary Computation*. PhD thesis, Instituto Superior Técnico, Lisbon, November 2004.