INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

**Área Departamental de Engenharia de
Eletrónica 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

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

Júri:
Presidente: Mestre Pedro Alexandre Seia Cunha Ribeiro Pereira
Vogais:
Doutor Artur Jorge Ferreira
Doutor Luís Filipe Graça Morgado

Outubro, 2015

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Eletrónica 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

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

Júri:
- Presidente: Mestre Pedro Alexandre Seia Cunha Ribeiro Pereira
- Vogais:
  - Doutor Artur Jorge Ferreira
  - Doutor Luís Filipe Graça Morgado

Outubro, 2015

Acknowledgments

I would like to express my gratitude to my supervisors Artur Ferreira and Nuno Leite, during the entire project development. If it wasn't for them, the development of this project wouldn't be possible. Their assistance and motivation really made me to continue and to work hard on this project.

I would like to thank my best friends João Vaz, Daniel Albuquerque, André Ferreira, Pedro Miguel, my brothers Pedro Nunes and João Tiago, my parents Mariana and Paulo, and my girlfriend Ana Sofia, for always being there to support me on the hardest moments of my life. Their help on my personal life issues was crucial while developing this project.

Last, but not the least, I would like to thank Instituto Superior de Engenharia de Lisboa, for giving me support and knowledge in the Computing Engineering subject, for the past six years.

Resumo

Nos últimos anos, o tema da geração automática de horários tem sido alvo de muito estudo. Em muitas instituições, a elaboração de horários ainda é feita manualmente, constituindo-se uma tarefa demorada e penosa para instâncias de grande dimensão. Outro problema recorrente na abordagem manual é a existência de falhas dada a dificuldade do processo de verificação, e também a qualidade final do horário produzido. Se este fosse criado por computador, o horário seria válido e seriam de esperar horários com qualidade superior dada a capacidade do computador para pesquisar o espaço de soluções.

A elaboração de horários não é uma tarefa fácil, mesmo para uma máquina. Por exemplo, horários escolares necessitam de seguir certas regras para que seja possível a criação de um horário válido. Mas como o espaço de estados (soluções) válidas é tão vasto, é impraticável criar um algoritmo que faça a enumeração completa de soluções a fim de escolher a melhor solução possível. Por outro lado, a utilização de algoritmos que realizam a enumeração implícita de soluções (por exemplo, branch and bound), não é viável para problemas de grande dimensão. A utilização de heurísticas que percorrem de uma forma guiada o espaço de estados, conseguindo assim uma solução razoável em tempo útil, constituem uma abordagem adequada para este tipo de problemas.

Um dos objetivos do projeto consiste na criação duma abordagem que siga as regras do *International Timetabling Competition (ITC) 2007* incidindo na criação de horários de exames em universidades (Examination timetabling track). Este projeto utiliza uma abordagem de heurísticas híbridas. Isto significa que utiliza múltiplas heurísticas para obter a melhor solução possível. Utiliza uma variação da heurística de Graph Coloring para obter uma solução válida e as meta-heurísticas Simulated Annealing e Hill Climbing para melhorar a solução obtida.

Os resultados finais são satisfatórios, pois em algumas instâncias os resultados são melhores do que alguns dos cinco finalistas do concurso ITC 2007.

Palavras Chave

Heurísticas; Meta-heurísticas; International Timetabling Competition 2007; Horários; Horários de exames; Coloração de Grafos; Solidificação Simulada;

Abstract

In the last few years the automatic creation of timetables is being a well-studied subject. In many institutions, the elaboration of timetables is still manual, thus being a time-consuming and difficulty task for large instances. Another current problem in the manual approach is the existence of failures given the difficulty in the process verification, and so the quality of the produced timetable. If this timetable had been created by a computer, the timetable would be valid and timetables with better quality should be obtained, given the computer's capacity to search the solution space.

It is not easy to elaborate timetables, even for a machine. For example, scholar/university timetables need to follow certain type of constraints or rules for them to be considered valid. But since the solution space is so vast, it is highly unlikely to create an algorithm that completely enumerates the solutions in order to choose the best solution possible, considering the problem structure. The use of algorithms that perform implicit enumeration solutions (for example, an branch bound), is not feasible for large problems. Hence the use of heuristics which navigate through the solution space in a guided way, obtaining then a reasonable solution in acceptable time.

One main objective of this project consists in creating an approach that follows the *International Timetabling Competition (ITC) 2007* rules, focusing on creating examination timetables. This project will use a hybrid approach. This means it will use an approach that includes multiple heuristics in order to find the best possible solution. This approach uses a variant of the Graph Coloring heuristic to find an initial valid solution, and the meta-heuristics Simulated Annealing and Hill Climbing to improve that solution.

The final results are satisfactory, as in some instances the obtained results beat the results of some of the five finalists from ITC 2007.

Keywords

Heuristics; Meta-heuristics; International Timetabling Competition 2007; Timetable; Examination Timetabling; Graph Coloring; Simulated Annealing;

Contents

Acknowledgments ... v

Resumo ... vii

Abstract ... ix

List of Figures ... xiii

List of Tables ... xv

List of Code Listings ... xvii

List of Acronyms ... xix

1 Introduction ... 1
- 1.1 Educational Timetabling Problems ... 1
- 1.2 Objectives ... 2
- 1.3 Document Organization ... 2

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 methods - further approaches ... 10

3 Software Architecture . . . 15
3.1 Data Layer . . . 15
3.2 Data Access Layer . . . 15
3.3 Business Layer . . . 17
3.4 Tools Layer . . . 19
3.5 Heuristics Layer . . . 20
3.6 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 . . . 21
4.2 Graph Coloring . . . 23
4.2.1 Implementation . . . 23
4.2.2 Stochasticity . . . 25
4.3 Solution Initialization Results . . . 25

5 Proposed Approach: Local Search . . . 29
5.1 Simulated Annealing . . . 29
5.1.1 Implementation . . . 30
5.1.2 Variable Rate Computation . . . 32
5.2 Hill Climbing . . . 34
5.2.1 Implementation . . . 34
5.3 Neighborhood Operators . . . 35
5.3.1 Implementation . . . 37
5.3.2 Statistics . . . 37
5.3.3 Neighborhood Operators Effect . . . 39
5.4 Fitness Computation . . . 40
5.4.1 Implementation . . . 40

6 Experimental Results . . . 43

7 Conclusions . . . 47
7.1 Future Work . . . 47

References . . . 49

List of Figures

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

3.1 Overview of the subsystems that compose the system software architecture. . . . 16
3.2 Overview of the DAL and the present entity types and repositories. 17
3.3 Overview of the BL and its main classes. 18

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 34
5.3 Hill Climbing classes 35
5.4 Neighborhood selection and operators 38
5.5 Accepted and rejected neighbors for each examination in the Simulated Annealing. 39

List of Tables

2.1 Timeline of existing approaches 14

4.1 Specifications of the 12 datasets of the ITC 2007 examination timetabling problem. 22
4.2 Some of the Graph Coloring's performance features. 27

5.1 Improvement factor (percentage) between using five and six neighbor operators. 40

6.1 Obtained results for the proposed SA hybrid algorithm. "–" indicates that a feasible solution could not be obtained. 44
6.2 Comparison of the fitness results of the current approach with the previous approach. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained. 44
6.3 Comparison of the proposed approach with the ITC 2007 finalists. The comparison is made between the average values of each approach. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained. 45
6.4 Comparison of the proposed approach with state-of-the-art approaches. The comparison is made between the average values of each approach. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained, or the following datasets were not tested. 45
6.5 Results with different time constraints. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained. 46

List of Algorithms

1 LoaderTimetabling's Load method. 23
2 Graph Coloring algorithm. 27
3 Simulated Annealing method. 32
4 Rate computing. 33

List of Acronyms

AI	Artificial Intelligence
BB	Branch and Bound
BL	Business Layer
CPBT	Constraint Programming Based Technique
CRUD	Create, Read, Update and Delete
DAL	Data Access Layer
DL	Data Layer
GA	Genetic Algorithm
GC	Graph Coloring
GD	Great Deluge
GRASP	Greedy Randomized Adaptive Search Procedure
HC	Hill Climbing
HL	Heuristics Layer
IFS	Iterative Forward Search
ILP	Integer Linear Problem
IP	Integer Programming
ITC 2007	International Timetabling Competition 2007
MILP	Mixed-Integer Linear Programming
PL	Presentation Layer
SA	Simulated Annealing
TL	Tools Layer
TS	Tabu Search

1

Introduction

Many people believe that Artificial Intelligence (AI) techniques aim 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

The educational timetabling problems involves 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 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, while 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 particular timetabling 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 Algorithm (GA) [2], Simulated Annealing (SA) [3], Tabu Search (TS) [4], and some of its hybridizations. The second phase consists in the development of the selected algorithms and promising hybridizations, and to 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 Document Organization

The remainder of this report is organized as follows. Chapter 2, addresses 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, algorithmic strategies to tackle the timetabling problem, and the methods of the five ITC 2007 finalists are summarized. In Chapter 3 the system architecture is explained, together with the created layers. Chapter 4 explains how the solution is created and initialized. Chapter 5 addresses the proposed local search meta-heuristics, along with their implementations. In

Chapter 6 compares the final results with the five ITC 2007 finalists and state-of-the-art approaches. Finally, in Chapter 7 we share the conclusions taken after developing the project, finishing with the possible future work.

2

State-of-the-Art

In this chapter, 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, sub-optimal solutions are feasible solutions with 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 [5]. 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 to find an initial feasible solution.

The basic examination timetabling problem, where only the *clash* hard constraint is observed, reduces to the well-known Graph Coloring (GC) [6] problem. The clash hard constraint specifies that no conflicting exams should be scheduled at the same time slot. Deciding whether a solution exists or not in the GC problem, is a NP-complete problem [7] [8]. Considering the GC as an optimization problem, it is proven that the task of finding the optimal solution is also a NP-Hard problem [7] [8]. GC problems are further explained in Section 2.2

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 [8]: *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 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 applied to obtain a solution, which may be feasible or not. For instance, GC 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 of problem. In these, one usually considers both hard and

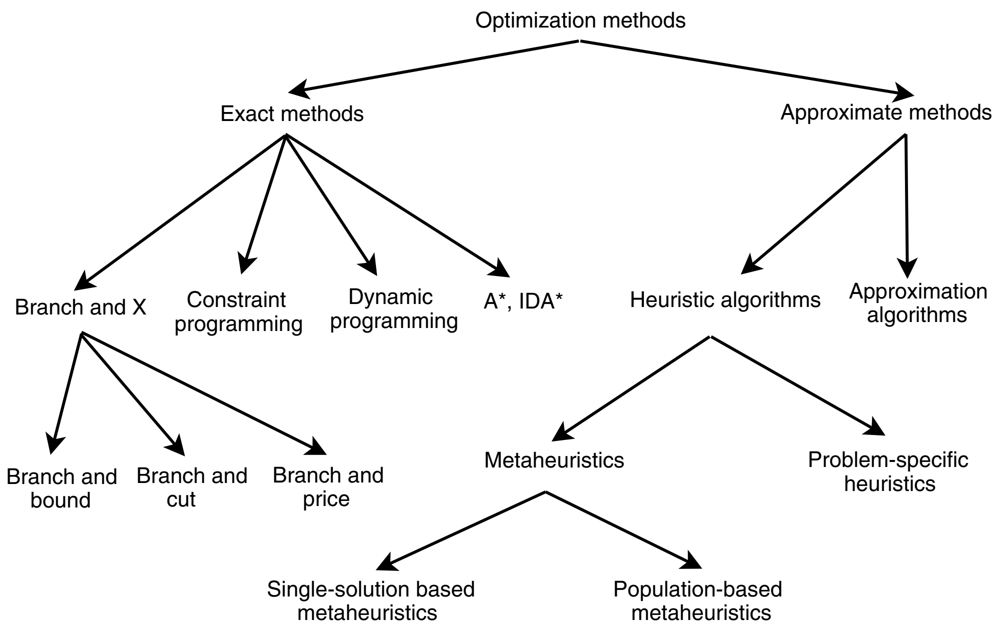

Figure 2.1: Optimization methods: taxonomy and organization (adapted from [1]).

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 [9]. The One-Stage algorithm is applied to get a solution, whose goal is to satisfy both hard and soft constraints, at the same time. The Two-Stage algorithms are the most frequent 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 that the optimal solution is found. 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 the large size problem instances, it may not be possible to use this approach due to the prohibitive time.

Constraint-Programming Based Technique

The Constraint Programming Based Technique (CPBT) [10] allows direct programming with constraints which gives ease and flexibility in solving problems like timetabling. Two important features of this technique are the use of backtracking and logical variables. Constraint programming is different from other types of programming, in the sense that it specifies 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 [8].

Integer Programming

The Integer Programming (IP) [11] is a mathematical programming technique in which the optimization problem to be solved must be formulated as an Integer Problem. If both the objective function and the constraints are 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 [5] 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 GC problem. Exploiting the connection between these two problems, several authors used two-phase algorithms in which GC heuristics are applied in the first phase, to obtain an initial feasible solution.

Graph Coloring Problem

The 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 GC 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 constraints are taken into consideration. Thus, soft constraints are ignored [8].

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 [12].

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, such as the GC heuristics. Meta-heuristics are divided in two main sub-types, 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 exploitation 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. The 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 computed by the 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.

in order to get a detailed description 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 [13].

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 [14] was actually applied to solve the 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 [15] to obtain feasible solutions and Conflict-based Statistics [16] to prevent IFS from looping. The second phase consists in using multiple optimization algorithms. These algorithms are applied in this order: Hill Climbing (HC) [17], Great Deluge (GD) [18], and optionally SA [3].

Gogos was able to reach second place in the Examination Timetabling track. Gogos' approach [19], like Müller's, is a two-phase approach. The first phase starts with a pre-processing stage, in which the hidden dependencies between exams are checked 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 applied only 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 force SA to restart with more promising solutions and to generate better results.

Atsuta et al. ended up in third place on the Examination Timetabling track and won third and second places on the other tracks, with the same approach for all of them. Their approach [20] consists on applying a constraint satisfaction problem solver adopting an hybridization of TS and Iterated Local Search.

Geoffrey De Smet's approach [4] 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* [21]. 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, namely TS, to improve the solutions obtained using the drools-solver.

Nelishia Pillay proposed a two-phase algorithm variant, using a *Developmental Approach based on Cell Biology* [22], whose 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 methods - further approaches

In this subsection, we describe other approaches to the ITC 2007 problem, which were proposed after the 2007 contest.

Abdullah et al.'s 2009 approach [23] consists on using an hybridization of an electromagnetic-like mechanism and the GD algorithm. In this approach, the electromagnetism-like mechanism starts with a randomly generated population of timetables. Electromagnetic-like mechanism is a meta-heuristic algorithm using an *attraction-repulsion* technique [24] to move the solutions to the region of optimal solutions.

McCollum et al.'s 2009 two phase approach [25] applies an adaptive ordering heuristic from [26], proceeding with an *extended version* of GD. As the author stated, this approach is robust and generic considering the obtained results on the benchmark datasets from ITC 2007.

Alzaqebah et al.'s 2011 two phase approach [27] starts by using a GC heuristic (largest degree ordering) to generate the initial solution. It ends by applying the *Artificial Bee Colony* search algorithm to optimize the solution.

Turabieh and Abdullah's 2011 approach [28] utilizes a two phase algorithm. The first phase consists on constructing initial solutions by using an hybridization of GC 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 [23].

Turabieh and Abdullah proposed another approach in 2012 [2] that utilizes a Tabu-based memetic algorithm which consists on an hybridization of a GA with TS algorithms. The authors claim that this approach produces some of the best results, when tested on the ITC 2007's datasets.

Demeester et al. in 2012 created an hyper-heuristic approach [29]. The heuristics that were considered are 'improved or equal', equivalent to hill climbing that accepts equally good solutions, SA, GD, and an adapted version of the *late acceptance* strategy [30]. 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 [31] introduce an IP formulation to the ITC 2007 instance, and also a solver using the CPLEX software.

Sabar et al.'s utilized a GC hyper-heuristic on its approach in 2012 [32]. This hyper-heuristic is composed of four hybridizations of these four methods: last degree, saturation degree, largest colored degree and largest enrollment. This approach compete with the winners' approaches from ITC 2007, considering the benchmark results.

Sabar et al.'s 2012 approach [33] utilizes a two phase algorithm. It starts by using an hybridization of GC 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 [34], mixing the use 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 [35] that utilizes an hybridization of a modified artificial bee colony with a local search algorithm (i.e. late acceptance HC).

Burke et al.'s 2014 approach [36] 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 [37] 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 the GD algorithm.

Hamilton-Bryce et al. [38] in 2014 opted to use a non-stochastic method on their approach [38], 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 into 3 phases. The first phase uses a *Squeaky Wheel* constructor which generates multiple initial timetables and a weighted list for each timetable. 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 a weighted list of values is calculated, based on the solution's individual soft constraints penalty.

Rahman et al.'s approach [39] 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 GC heuristics like largest degree and saturation degree heuristics.

Rahman et al.'s approach [40] utilizes adaptive linear combinations of graph coloring heuristics (largest degree and saturation degree) with an heuristic modifier. These adaptive linear combinations allow 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 contains a timeline that represents the previous mentioned approaches chronologically ordered.

Table 2.1 Timeline of existing approaches

2007	Atsuta et al. - Constraint satisfaction problem solver using an hybridization of TS and Iterated 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 in 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 GC 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.
	Sabar et al. - GC 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 such as late acceptance HC.
	Burke et al. - Uses an hyper-heuristic with hybridization of low level heuristics (neighbor operations), 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 utilizes an hybridization of HC, SA and an extended version of the 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. - Utilizes adaptive linear combinations of GC heuristics, like largest degree and saturation degree, with heuristic modifier.

3

Software Architecture

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

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.

This project was developed using .NET (C# programming language). It was implemented using Microsoft Visual Studio Community 2013 as the development tool, on Microsoft Windows 7 Professional Operating System.

3.1 Data Layer

The DL stores all entities. 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.

3.2 Data Access Layer

The DAL allows 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 the `IEntity` interface. 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 their respective identifiers. The `Repository` provides basic Create, Read, Update and Delete (CRUD)

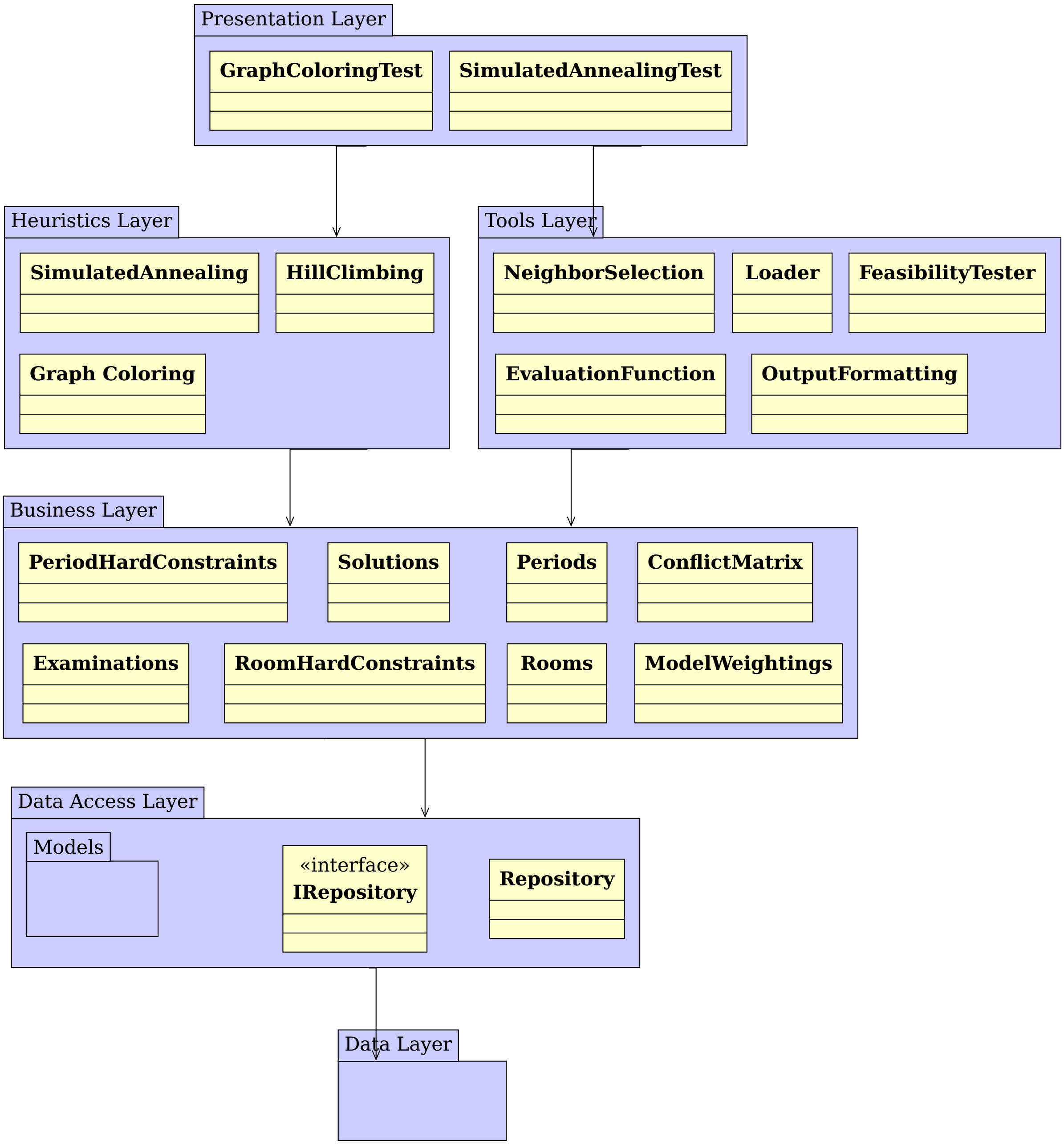

Figure 3.1: Overview of the subsystems that compose the system software architecture.

functions to access and edit the stored entities. The signature of the entities and all the specifications of the generic repository are presented in Figure 3.2.

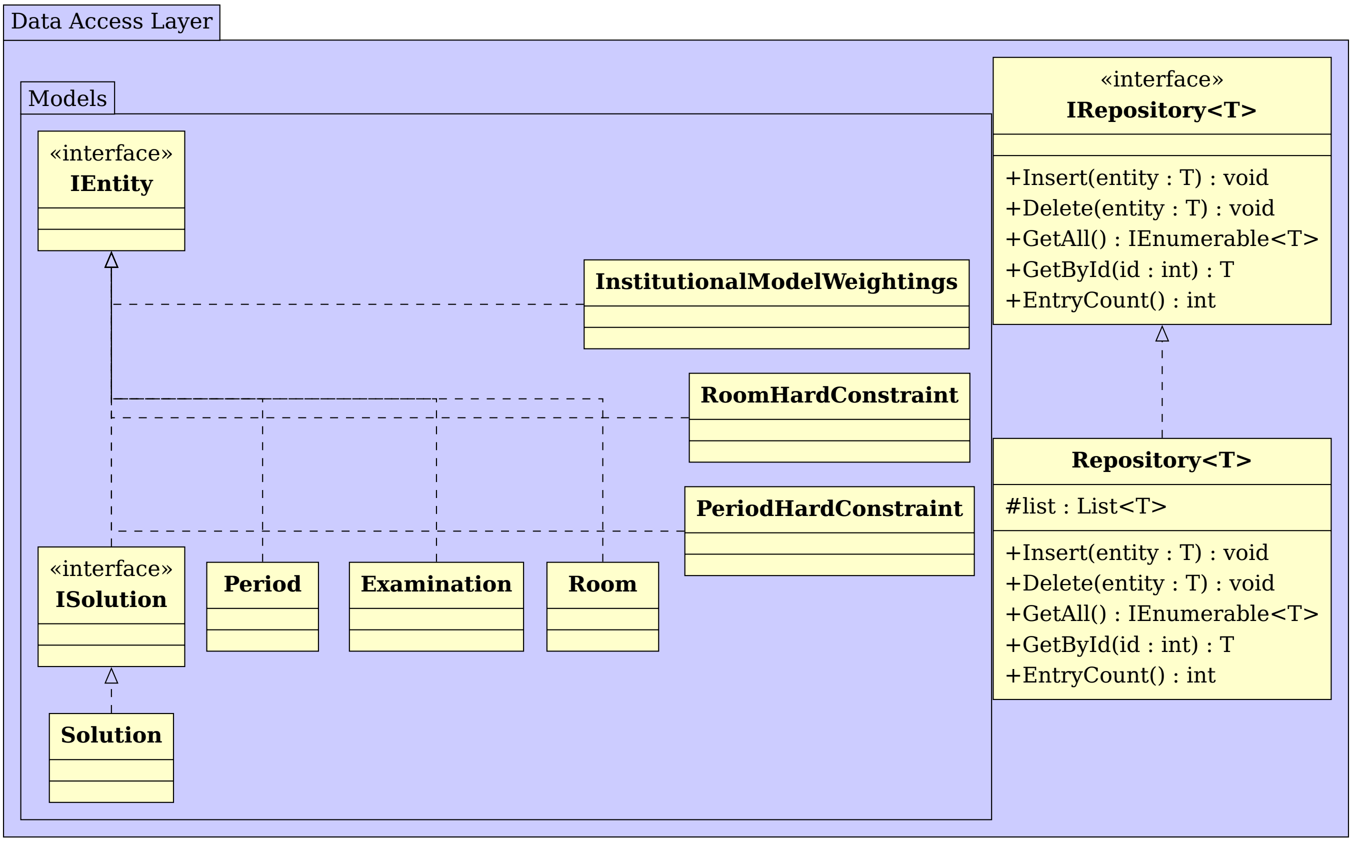

Figure 3.2: Overview of the DAL and the present entity types and repositories.

3.3 Business Layer

The BL allows access to the repositories explained above by, in each of the business classes, providing 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 hard constraints of a given type. This can be seen in Figure 3.3, which includes all the BL classes, methods, and variables. The CRUD functions provided by some of the business classes 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, using 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; instead, they provide *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

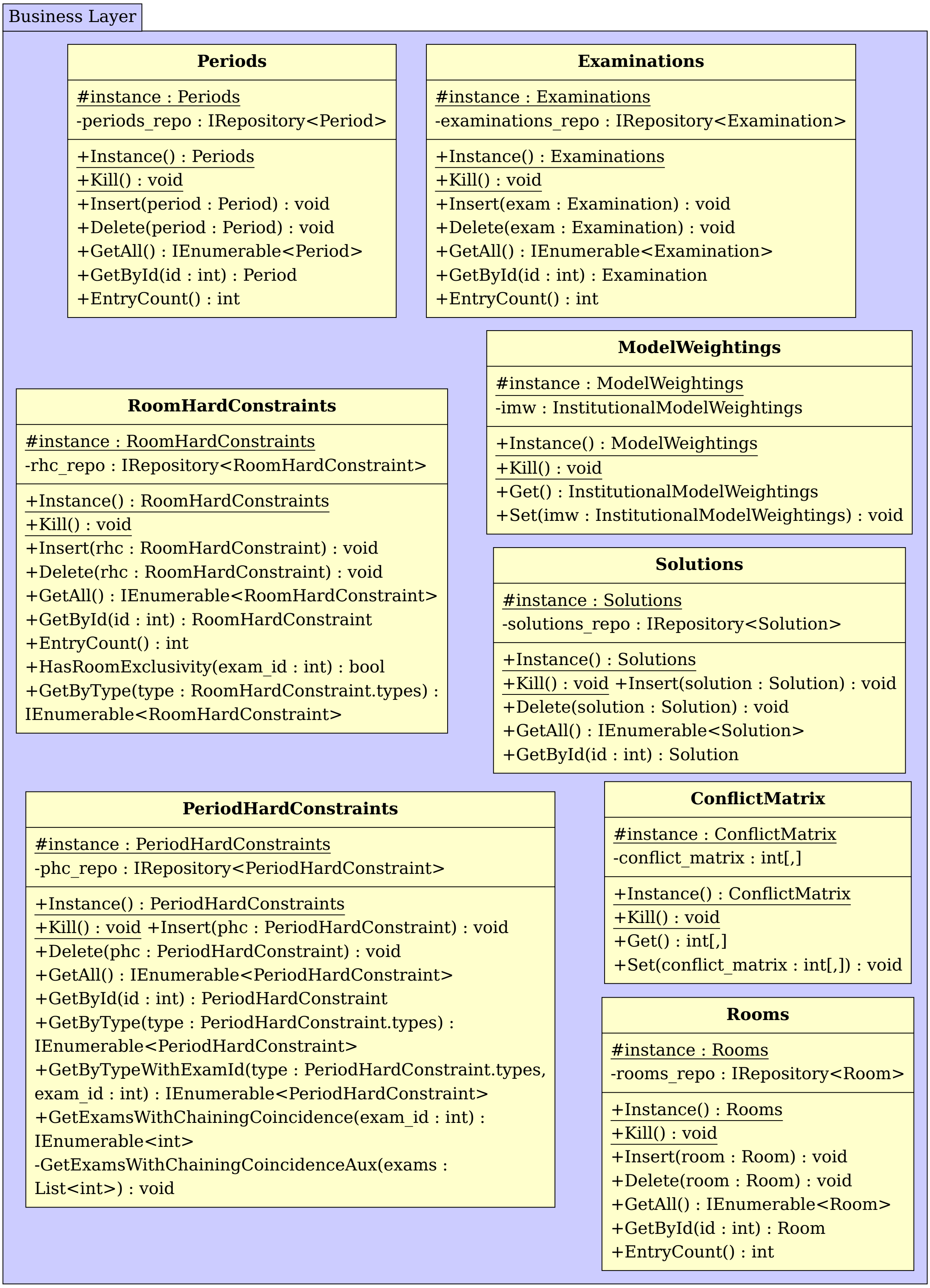

Figure 3.3: Overview of the BL and its main classes.

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 dataset. Each dataset must be loaded using the `Loader` if that dataset 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 it must be populated using the Loader each time a dataset 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 easily be accessed statically.

3.4 Tools Layer

The TL contains all the tools used by the HL and by the lower layers, while using the BL to access 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. The fitness value refers to the score of the solution depending on the violated soft constraints and their penalty values. The distance to feasibility is used by the GC heuristic to guarantee that the end solution is feasible, while the fitness is applied by the meta-heuristics, 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 is given in 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 are given in Section 5.3.

The `FeasibilityTester` is a tool that provides functions, which efficiently check if a certain examination can be placed in a given 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 provided solution.

The `OutputFormatting` tool is used to create the output file, given the final solution. This file obeys the output file rules of the ITC 2007's site [41], in order to be able to submit the solution [42]. Submitting the solution allows one to check all violated hard constraints, soft constraints, distance to feasibility and fitness values on the site's page.

3.5 Heuristics Layer

The HL offers access to all the implemented heuristics. These are the GC, SA, and HC. All these heuristics are applied to create the best timetable, with time constraints. Heuristics like SA and HC make use of neighbor solutions, and so they 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.

A detailed explanation about these heuristics is provided in both Chapters 4 and 5.

3.6 Presentation Layer

The PL's objective is to run the project functionalities, invoking all the other modules, and to check the final results. It's in this layer that all the experiments and tests are made, such as changing input parameters on both SA and HC, to check if better results can be achieved.

4

Loader and Solution Initialization

The Loader and the solution initializer (heuristic) are the first tools used in this project. The Loader and Graph Coloring heuristic (solution initializer) are executed only once, so the major part of the execution time will be used by the meta-heuristic(s).

4.1 Loader Module

The Loader's job is to load all the information presented in the dataset files. Each dataset file includes information about examinations, enrolled students, periods, rooms and their penalties, period and 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 their corresponding repositories, but also creates and populates the conflict matrix depending on the data obtained previously. The conflict matrix contains the information about the conflicts between each pair of examinations.

4.1.1 Analysis of benchmark data

Table 4.1 presents the specifications of the 12 datasets of the ITC 2007. The instances presented have different degrees of complexity, that is, ease of finding a feasible solution and number of feasible solutions. Using the developed GC heuristic (described in Section 4.2), dataset 4 is the most complex one, as no feasible solution could be obtained. The high conflict matrix density, in addition to the fact there's only one room (with the capacity of 1200 seats), and the presence of 40 period hard constraints, help to explain the dataset complexity.

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

4.1.2 Implementation

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

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

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

The loader was implemented using the `Loader` base class, from which the `LoaderTimetable` extends. The `Loader` class provides methods to read the file contents. These two classes are depicted in Figure 4.1.

Unlike the `Loader`, `LoaderTimetable` depends on the structure of the dataset files. This class will use the Loader functions to go through a dataset file, and so, populate the repositories depending on the information read by the `Loader` class. The `LoaderTimetable` offers operations like `Load` and `Unload` to load all the information in a dataset file and to empty the repositories, respectively.

The implementation of the `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 consists in a cycle that gets a new line and checks if the first string (e.g., “Exams”) is contained on that line. If so, it runs `InitExaminations`, if not, checks if the next string (e.g. “Periods”) is contained on that line, and so on, until it reaches the end of the file. The pseudo code of this method can be seen on Algorithm 1.

The method `InitExaminations` reads each examination and enrolled students, and stores them in a Hashmap, treating the students as keys and the examinations which they attend as values. With this student-examinations organization, the `InitConflictMatrix` simply goes to all pairs of examinations for each student, and add a conflict in the matrix for that pair of examinations.

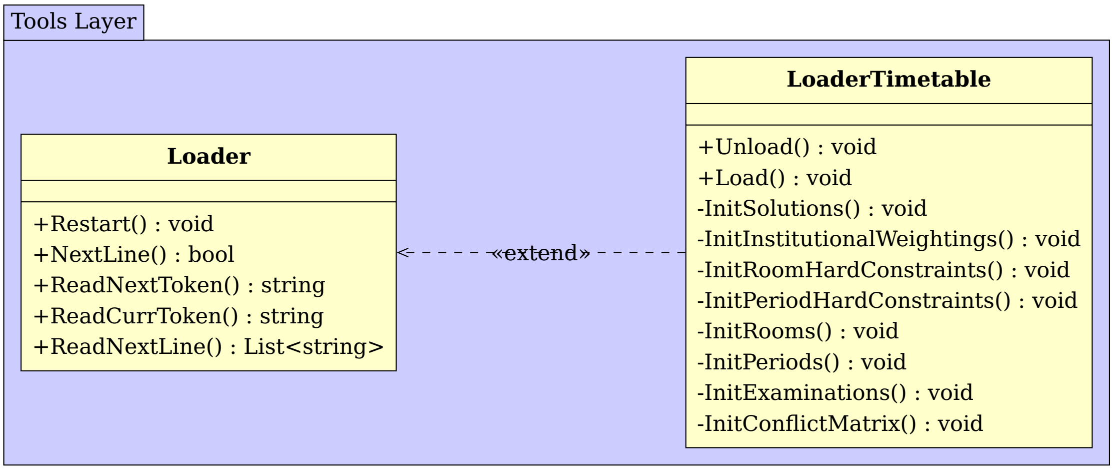

Figure 4.1: Specification of Loader and LoaderTimetable tools

4.2 Graph Coloring

Graph Coloring is the heuristic used to generate a feasible solution. This heuristic runs right after the loader. The implementation of this heuristic was based on Müller's approach [14].

4.2.1 Implementation

The computation of the proposed heuristic is divided into four 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

Algorithm 1 LoaderTimetabling's Load method.

Input: *void*
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
$InitSolutions()$
$InitConflictMatrix()$
Output: *void*

the student conflicts and exclusion.

The second phase erases all examination coincidence hard constraints' occurrences that have student conflicts. It is mentioned in the ITC 2007 website [43] that if two examinations have the examination coincidence hard constraint and yet 'clash' with each other due to student enrollment, this hard constraint is ignored.

The third phase populates and sort the assignment lists, which are four lists that hold 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 Graph Coloring heuristic is used on these four lists, and so, each list is sorted by the number of student conflicts. The examination assignment is done taking the present list ordering. First, all the examinations with room exclusivity are assigned, then all that have the after constraints, and finally all with examination coincidence.

The fourth phase is the examination assignment phase (the most important phase of this heuristic). It is based on Müller's approach [14]. 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 will not occur. Instead, only that period will be considered and if the period is not feasible to the current examination, the normal assignment will not take place.

- Forcing assignment – Occurs if there are no possible periods to assign the chosen examination, because 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. Forcing an examination to a specific period because of a coincident examination sometimes caused infinite or very long loops in dataset 6. Because of this, there's a chance of 25% that the previous rule does not occur, but instead, it unassigns all coincident examinations and try to assign the current examination to a random period. We chose 25% because this event must happen rarely to avoid infinite loops. *Exam_coincidence* hard constraint is present in larger amounts only in a few datasets, and is only applied to pairs of examinations. Finding an *exam_coincidence* chain with more than 3 examinations is very rare, so even if this event triggers in other datasets, it is not considered a problem. It would be considered a problem if the event was to be triggered more often, or if bigger *exam_coincidence* chains existed, degrading the efficiency of the heuristic.

The GC heuristic implementation only performs assignments and unassignments of examinations and checks examination clashes, when forcing an assignment. The feasibility checking done to periods and rooms is made by the `FeasibilityTester` tool, as mentioned in Section 3.3. The `GraphColoring` and `FeasibilityTester` classes, with all their attributes and methods, can be seen in Figure 4.2. The algorithm of the GC heuristic, which was explained in this section, is described in Algorithm 2.

4.2.2 Stochasticity

In order to produce random initial solutions, the implemented GC heuristic must have a stochastic behavior, i.e., it should produce different results in different runs. To verify the GC heuristic stochasticity, the following test was conducted. In order to have a statistically significant result, a total of 10 runs were performed; for each run, the period where each exam was scheduled was registered. Finally, it was observed that the periods distribution was random meaning that the generated timetables have the same examinations placed in random periods for different runs, forming random solutions.

4.3 Solution Initialization Results

The goal of the initialization procedure is not to find the best possible solution (in terms of fitness), but find feasible solutions. Some datasets are simpler than others, always getting good and stable execution times. Others can be harder, getting worse execution times and instability. An unstable dataset means that 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 datasets can be seen in Table 4.2. These results were obtained by running the heuristic, 10 times for each dataset, and computing the average for both fitness and execution time.

Table 4.2 exhibits the average fitness of the solutions created for each dataset, the average execution time and standard deviation, and the percentage of time used in the first and second phase of this project. The standard deviation was taken due to the large variation on execution times of datasets when creating initial solutions, using the GC heuristic. Observing the results, it can be concluded that datasets 6, 11, and 12, present some degree of instability due to a large execution time standard deviation, compared to the other datasets. The datasets 11 and 12, even though the average execution time was 3405 and 3690, the tested values varied from 864 to 7642 and from 201 to 6452, respectively.

The allowed execution time is 221000 milliseconds for each test made to each dataset. This time was picked by the benchmark program provided by the ITC 2007 site [44] when executed on the machine where all testes were performed. In the results presented on Table

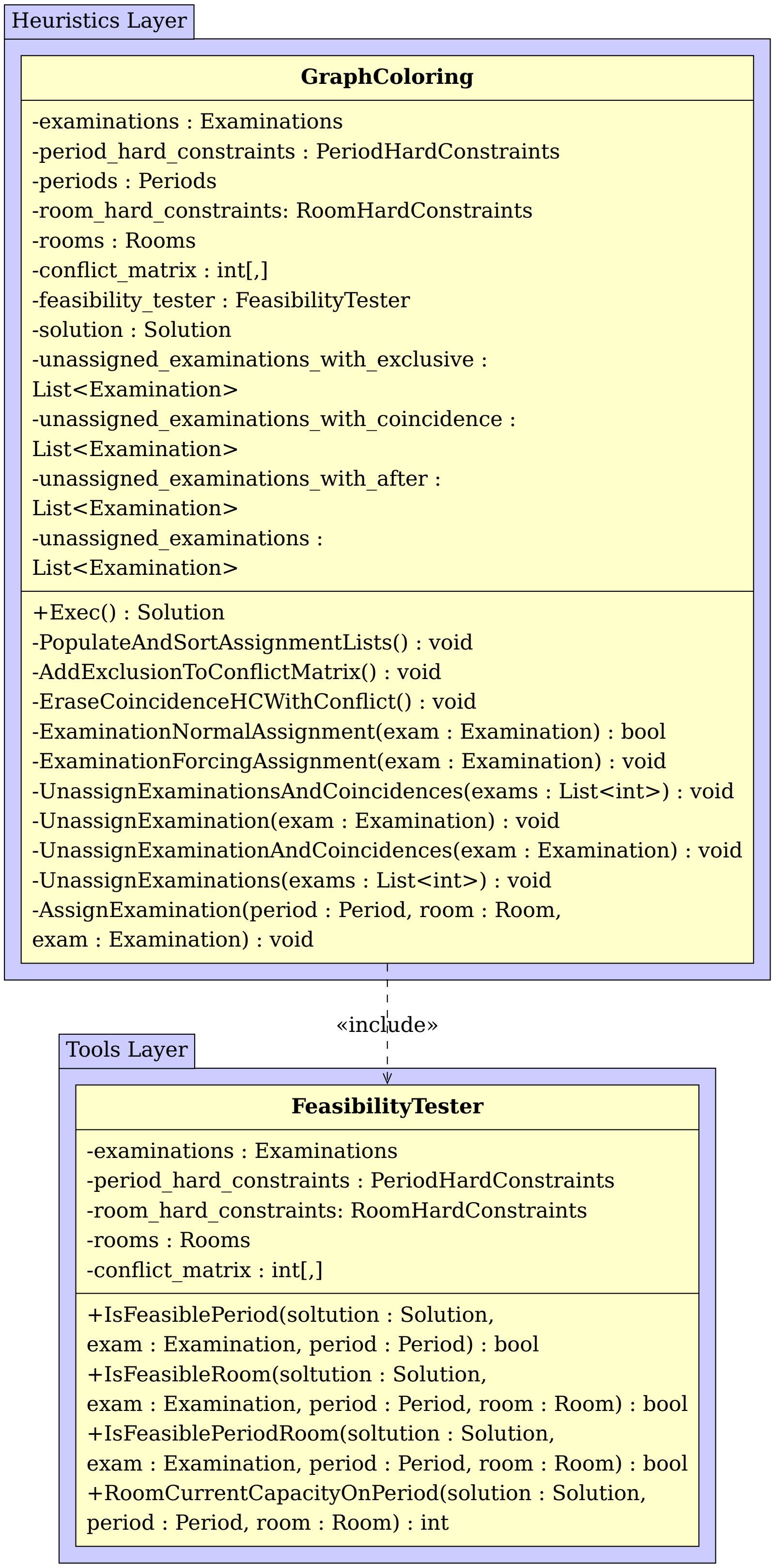

Figure 4.2: Graph Coloring and Feasibility Tester

Algorithm 2 Graph Coloring algorithm.

- Initial solution $solution$

```
Input: void
Add exclusion to conflict matrix
Erase coincidence HC that contains conflict HC
Populate and sort assignment examination lists
repeat
    Get the right list to use list
    Remove last examination from list, exam
    If Not NormalAssign(solution, exam) Then ForceAssign(solution, exam)
until No more examinations to assign
Output: solution
```

Table 4.2: Some of the Graph Coloring's performance features.

Dataset	Fitness	Execution time (ms)	Standard Deviation σ (ms)	Time elapsed in the 1_{st} phase (%)	Time elapsed in the 2_{nd} phase (%)
1	49 028	202	61	0.09	99.91
2	103 907	250	2	0.11	99.89
3	170 232	942	20	0.43	99.57
4	–	–	–	–	–
5	349 319	227	5	0.10	99.9
6	60 857	297	192	0.13	99.87
7	155 708	569	19	0.26	99.74
8	397 868	243	8	0.11	99.89
9	15 680	12	1	0.01	99.99
10	121 164	110	6	0.05	99.95
11	260 310	3405	2170	1.54	98.46
12	11 887	3690	1829	1.67	98.33

4.2, we can conclude that a very short amount of time is used on this first phase, as compared to the one being used on the second phase.

As shown in Table 4.2, the heuristic was not able to produce a feasible solution for dataset 4. The reason for this might be the presence of an infinite cycle of normal and forcing assignments, resulting in assigning and unassigning the same examinations. Different approaches have been tried in order to solve this problem. These include using a single unsorted list of unassigned examinations, using all four unsorted lists, and keeping the four lists sorted but changing the priority order of their usage. Oddly, using only one unsorted list led to the best results, sometimes leaving only 20 to 30 examinations to be placed, while the default approach varied between 90 and 120. Unfortunately none of the approaches were able to generate a feasible solution.

In the proposed approach, the exams were sorted by the number of students conflicts with other exams. Other approaches to tackle the problem would be, for example, use the same sorting criteria with examination conflicts.

5

Proposed Approach: Local Search

The proposed solution consists on the use of a local search meta-heuristic(s) in order to improve the solution given by the GC heuristic. This approach uses SA, based on Müller's approach [14], followed by the use of HC. The use of HC is justified by the fact that SA does not guarantee a good control of the execution time, and so the parameters are given to make it almost use all the available time, which is, as mentioned in the previous chapter, 221 seconds. The rest of the optimization is carried out by HC, whose execution time is perfectly controllable.

5.1 Simulated Annealing

SA 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 an acceptance criterion. A neighbor solution is obtained by applying a movement operator (also known as neighbor operator) to the current solution, creating a new solution which is a neighbor of the current one. A neighbor operator, in this context, could be the movement of an examination to another time slot. Being a single-solution meta-heuristic, it generates a single neighbor. The neighbor operators and the acceptance criterion are the most important parts of this algorithm. Changes on one of these, may get the algorithm to behave in very different ways and end up with quite different solutions.

The acceptance criterion will, considering the current and a neighbor solution, give the percentage of acceptance of the neighbor solution. Most of the approaches using this heuristic accept a new neighbor solution if it is *better* than the current one. Otherwise, there's a chance that the neighbor solution is still accepted, depending on certain parameters. These parameters 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 at a given rate. The SA algorithm finishes when the current temperature is lower or equal to the minimum temperature. The temperature should start high enough in order to be able to escape from local optima, by accepting worse solutions found during the trajectory.

5.1.1 Implementation

The SA was implemented in a way that it is independent from the type of the cooling schedule and neighbor generator. The `SimulatedAnnealing` 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 `SimulatedAnnealingTimetable` implements the `SimulatedAnnealing` abstract class, which implements the neighbor generator method and the evaluation function attribute. The `SimulatedAnnealing` and `SimulatedAnnealingTimetable`'s methods and attributes can be seen in Figure 5.1.

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

The `ExecLinearTimer` has a linear cooling schedule, which is proportional to the running time, and uses the acceptance criterion

$$P(\delta E, T) = e^{\frac{-\delta E}{T}}, \tag{5.1}$$

with T being the current temperature and δE the fitness difference between the new neighbor and current solution.

The `Exec` method shares the same acceptance criterion but uses a geometric cooling schedule

$$T(i+1) = T(i).r, \tag{5.2}$$

with r being the temperature decreasing rate (0 < r < 1). In the geometric cooling schedule, the closer the rate is to unity, 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 [45]

$$T = T_{max}e^{-r.t}, \tag{5.3}$$

where t is the current span (counter initiated at 0), T_{max} the maximum/initial temperature, and r the decreasing rate.

This method also uses a different acceptance criterion

$$P(\delta E, T, f(s)) = e^{\frac{-\delta E}{T.f(s)}}, \tag{5.4}$$

being δE the fitness difference between the new neighbor and current solution, T the current temperature, and f(s) the solution fitness.

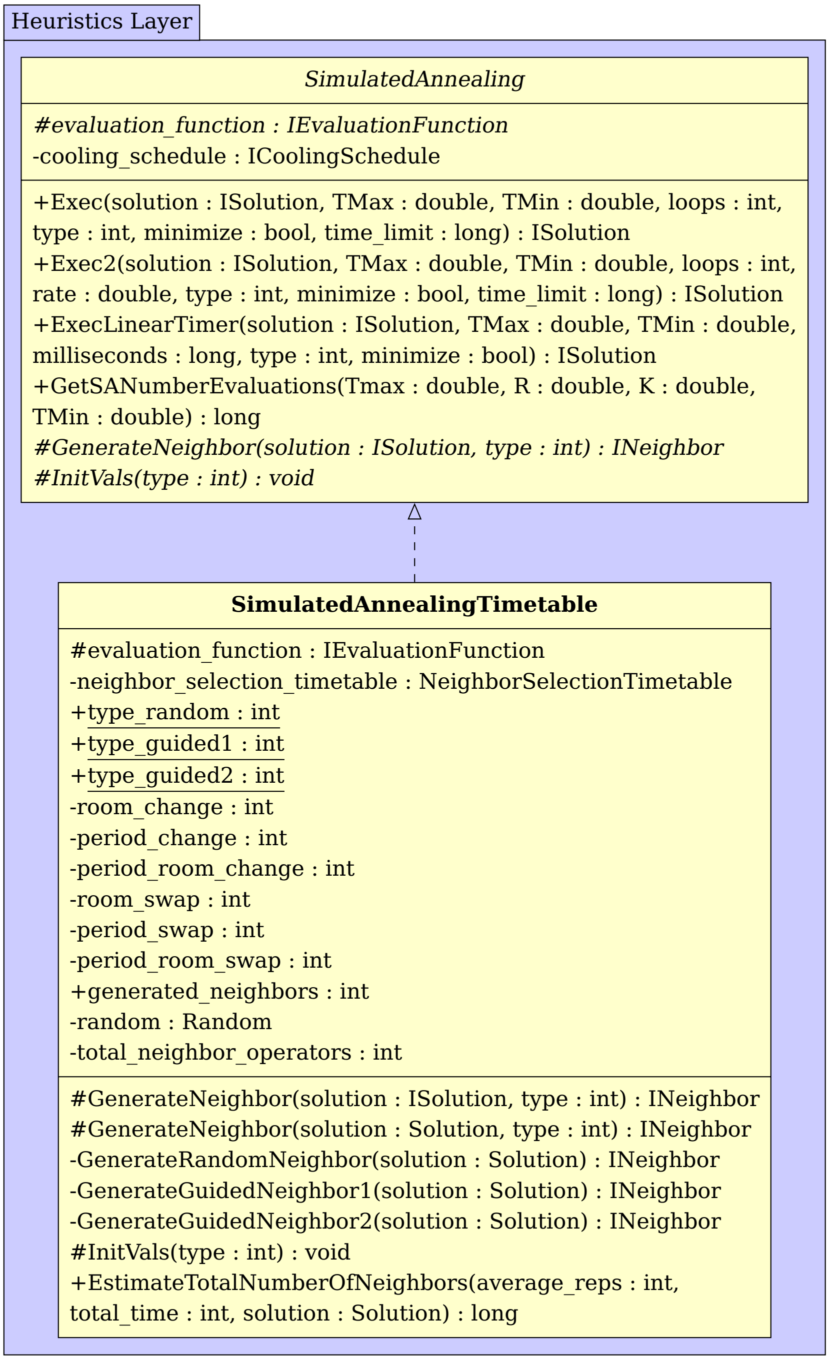

Figure 5.1: Simulated Annealing classes

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

5.1.2 Variable Rate Computation

As mentioned previously in Section 4.3, according to the ITC 2007 rules, the allowed time considering the used computer is 221000 milliseconds. As this time limit is the only imposition, it was decided to implement a SA with an adaptive cooling schedule, in order to use all the available time for the optimization process independently of the chosen dataset. This approach contrasts with the one in which a fixed cooling schedule is used. Each dataset has its own characteristics and using a given set of parameters for one set does not guarantee that the results will be equivalently good for the other datasets. For example, suppose that we've determined the best parameters for the first dataset, considering a time limit of 200000 milliseconds. Running the algorithm using the same parameters for the remaining datasets will not terminate on the time limit of 200000 milliseconds: the datasets with less number of resources (exams, rooms, students) will be optimized using less time; on the other way, the largest datasets will demand more time, eventually running over the imposed time limit.

Hence, a SA with a variable rate was implemented to make this heuristic run closer to the given time limit for all the datasets. Considering it is not certain that the algorithm will run within the given 221000 milliseconds, because of its stochastic nature, a time limit was added to this meta-heuristic as well, ending this heuristic automatically if the time limit is reached. To avoid the performance degradation incurred by the algorithm's halting before reaching the end of the optimization, a time offset was imposed. This offset is a percentage of the total allowed algorithm execution time. For example, instead of letting the heuristic run for 221000 milliseconds, one allows it to run for 185640 milliseconds, to have a safety margin. The offset used is 16%.

In order to determine the proper cooling schedule, the algorithm starts by simulating the execution of this heuristic, by running all the neighbor operators $AverageReps$ times, which

Algorithm 4 Rate computing.

Input:

- s // Solution
- $TMax$ // Maximum temperature
- $TMin$ // Minimum temperature
- $reps$ // Number of loops per temperature
- $exec\_time$ // Execution time limit

$sa = SAInit()$;
$n = 50$; // Number of times each operator runs
$comp\_neighbors = sa.EstimateNumberNeighbors(n, exec\_time, s)$;
$rate = 1.50^{-02}$; // Initial default rate
$power = -3$;
$rate\_to\_sub = 10^{power}$; // Rate decrementing
$total\_neighbors = sa.GetNumberNeighbors(TMax, rate, reps, TMin)$;
repeat
  If $rate - rate\_to\_sub \leq 0$ **Then** $power = power - 1$; $rate\_to\_sub = 10^{power}$;
  $rate = rate - rate\_to\_sub$;
  $total\_neighbors = sa.GetNumberNeighbors(TMax, rate, reps, TMin)$;
until $total\_neighbors < comp\_neighbors$
$rate = rate + rate\_to\_sub$; // To guarantee that total_neighbors < comp_neighbors
Output: $rate$ // Return computed rate

in this approach, the $AverageReps$ used is 50. Using the elapsed time and the given time limit, we estimate the number of neighbors that would be generated if this heuristic was to be run within the time limit. In order to compute the number of total neighbors, we use the expression

$$CompNeighbs = TotalTime/CompTime * AverageReps * TotalOperats, \quad (5.5)$$

being $TotalTime$ the time limit (221000 milliseconds), $CompTime$ the elapsed time used in the simulation, $AverageReps$ the number of loops that all the neighbors must run (50 runs), and $TotalOperats$ the total number of used operators (5 operators). After that, we compute the exact number of the neighbors ($TotalNeighbs$) that will be generated for the given parameters, using the default rate. This is achieved by simulating this heuristic using those parameters, cooling the current temperature until it reaches the minimum and returning the number of desired generated neighbors. If the $TotalNeighbs$ is above the $CompNeighbs$, a lower rate will be used to make another simulation, until the $TotalNeighbs$ for the given rate reaches a value that is close to the $TotalNeighbs$. The pseudo code of this method can be seen in Algorithm 4.

Some testings were made in order to check this heuristic's behavior, using the following parameters: *TMax* = 0.1, *TMin* = $1e - 06$, *loops* = 5, *exec_time* = 50000, and the computed_rate = 0.00016. As can be seen in Figure 5.2, it starts by accepting all worse and better neighbor solutions. In the end, the temperature is so low that it becomes harder to accept worse solutions, ending up acting similar to the HC procedure. The axis of abscissas represents the indexes of the generated neighbors and the axis of ordinates represents the current solution's score.

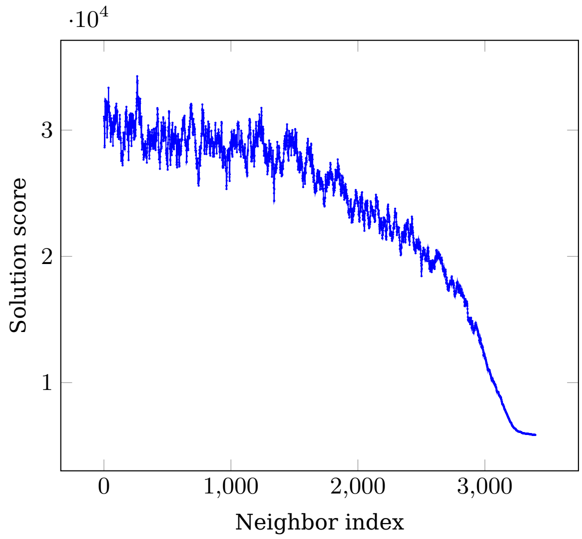

Figure 5.2: Simulated Annealing results

5.2 Hill Climbing

HC is a meta-heuristic different from the SA family of meta-heuristics, in the sense that it only accepts better solutions. So, as long as it reaches a local optimum, it can't get out of that point because each neighbor solution is worse. In this way, HC only has one parameter that is the number of iterations or time limit, which controls the algorithm's execution time.

In the evaluation undertaken, the best results were obtained with the `Exec2` version of SA. SA was parametrized in order to finish execution within the specified time limit imposed by the ITC 2007 rules. HC is executed after SA, using the remaining time, until the time limit is reached.

5.2.1 Implementation

The implementation of this heuristic is very similar to that of the SA; it contains the classes `HillClimbing` and `HillClimbingTimetable`, which are simplified versions of the SA classes. The `HillClimbing` and `HillClimbingTimetable`'s methods and attributes can be seen in Figure 5.3.

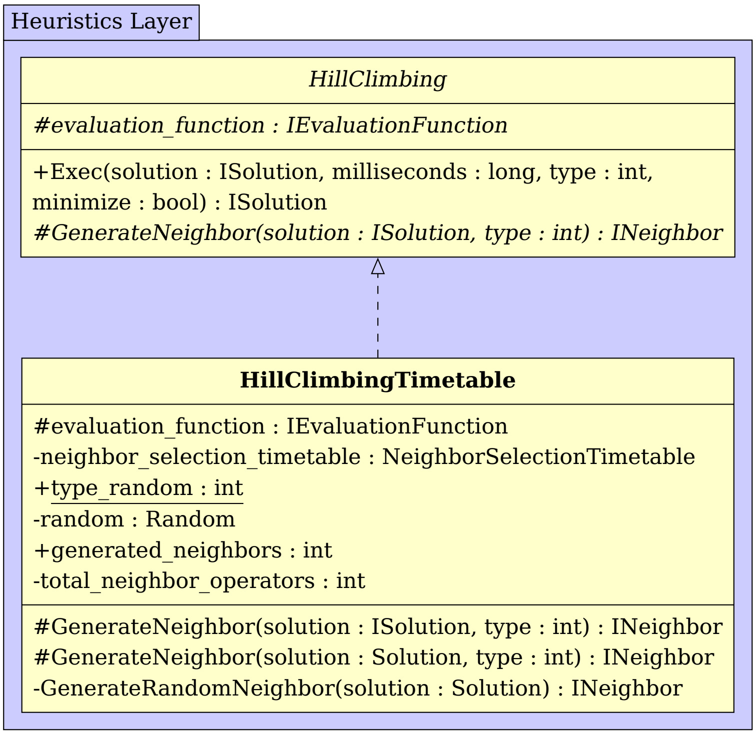

Figure 5.3: Hill Climbing classes

5.3 Neighborhood Operators

Neighborhood operators are applied 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 different approaches. Firstly, random selection was implemented. This approach always chooses a random operator to generate a new neighbor. Thereafter, 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 authors have implemented other variations of the first approach, where increasingly higher/lower probabilities of neighbor selection were defined based on the success/failure of the operator. Despite this, the random approach almost always showed better results compared to the guided approaches.

The neighborhood operators, in this context, are all based on moving examinations in terms of period or room. This implementation uses six different neighborhood operators:

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

- *Period Change* - An examination is randomly selected. After that, a period is randomly selected. If the assignment of the examination to the period, while keeping the room, does not clash with any hard constraints, that neighbor is returned. If not, the adjacent periods are sequentially checked until one of them creates a feasible solution. If it reaches the limit of periods and no feasible solution is found, no neighbor is returned;

- *Period & Room Change* - An examination is randomly selected. After that, a room and period are randomly selected. If the assignment of the examination to the room and period does not clash with any hard constraints, that neighbor is returned. If not, the adjacent rooms are checked 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 is found, no neighbor is returned;

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

- *Period Swap* - An examination is randomly selected. After that, a period is randonly 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 period, keeping the same room, does not clash with any hard constraints, that neighbor is returned. If not, the examinations presented in the adjacent 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 is found, no neighbor is returned;

- *Period & Room Swap* - An examination is randomly selected. After that, a period and room are randomly selected. If the selected examination can be placed in that period and room, then a *Period & Room Change* neighbor is returned instead. If not, if the swapping of the random examination with any of the examinations presented in the period and room does not clash with any hard constraints, that neighbor is returned. If not, the examinations presented in the adjacent 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 is 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. In order to have an efficient implementation, the neighbor only keeps the changes introduced in the solution, and not the changed solution itself. With this design, there's no need to replicate the original solution and to apply the neighborhood operator to it in order to obtain the neighbor solution. The process of replacing the original solution with the neighbor, consists in applying to the current solution the movement registered in the neighbor.

Every neighbor object must implement the interface `INeighbor`, which exposes the methods `Accept`, `Reverse` and a real value that represents the fitness of the neighbor (the fitness of the new solution if this neighbor is to be accepted). The `Accept` method moves the current solution to the neighbor solution. The `Reverse` method undoes the operation, getting then back the old solution. The different neighbors and their methods are illustrated in Figure 5.4.

5.3.2 Statistics

It is important to make sure that the algorithm works as planned. This includes the desired stochastic behavior, and all the neighbor operators must contribute positively to obtain better results.

In this particular case, to make sure this heuristic is stochastic, we must guarantee that all the examinations are moved roughly the same number of times. Considering that some examinations are harder to move, these are moved less times than others, but the difference is not significant. A study was made to compare, in the same run, the number of rejected and accepted neighbors for each examination. The examinations are sorted in descending order by the number of conflicts. Figure 5.5 represents a color map in which the colors represent the number of times each examination was accepted (x = 1) and rejected (x = 2), for dataset 1. As can be seen, some examinations are not even selected to move, and so, have zero accepted and rejected results. In this particular dataset, only two lines have zero counters, as checked in the results file. These two examinations have *exam_coincidence* hard constraint and are only able to be in one room due to the number of students being too high. The SA algorithm is not optimized to move more than one examination at the same time. This means that those two examinations can never be moved to another period, because the SA can only move one examination at a time and these examinations can't be moved to another room, because they don't fit in other rooms.

Other blue lines, which have values between 1000 and 1, are rare. This occurs not only because the examination is hard to move due to its conflicts, but may also have the *exam_coincidence* and a really short number of rooms in which that examination can fit. This problem creates a limitation in the SA heuristic, preventing the SA from scanning parts of the solution space, and eventually degrading the quality of the final result.

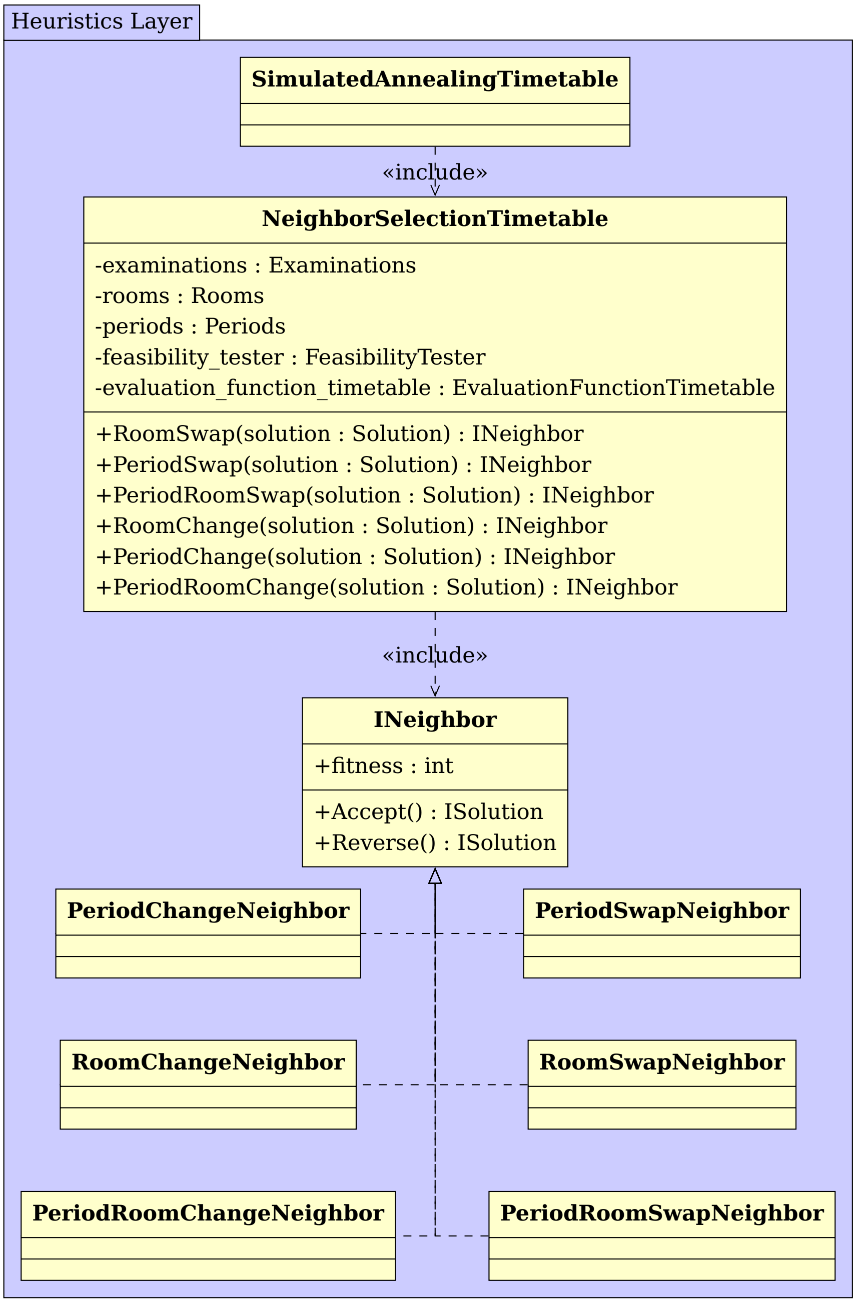

Figure 5.4: Neighborhood selection and operators

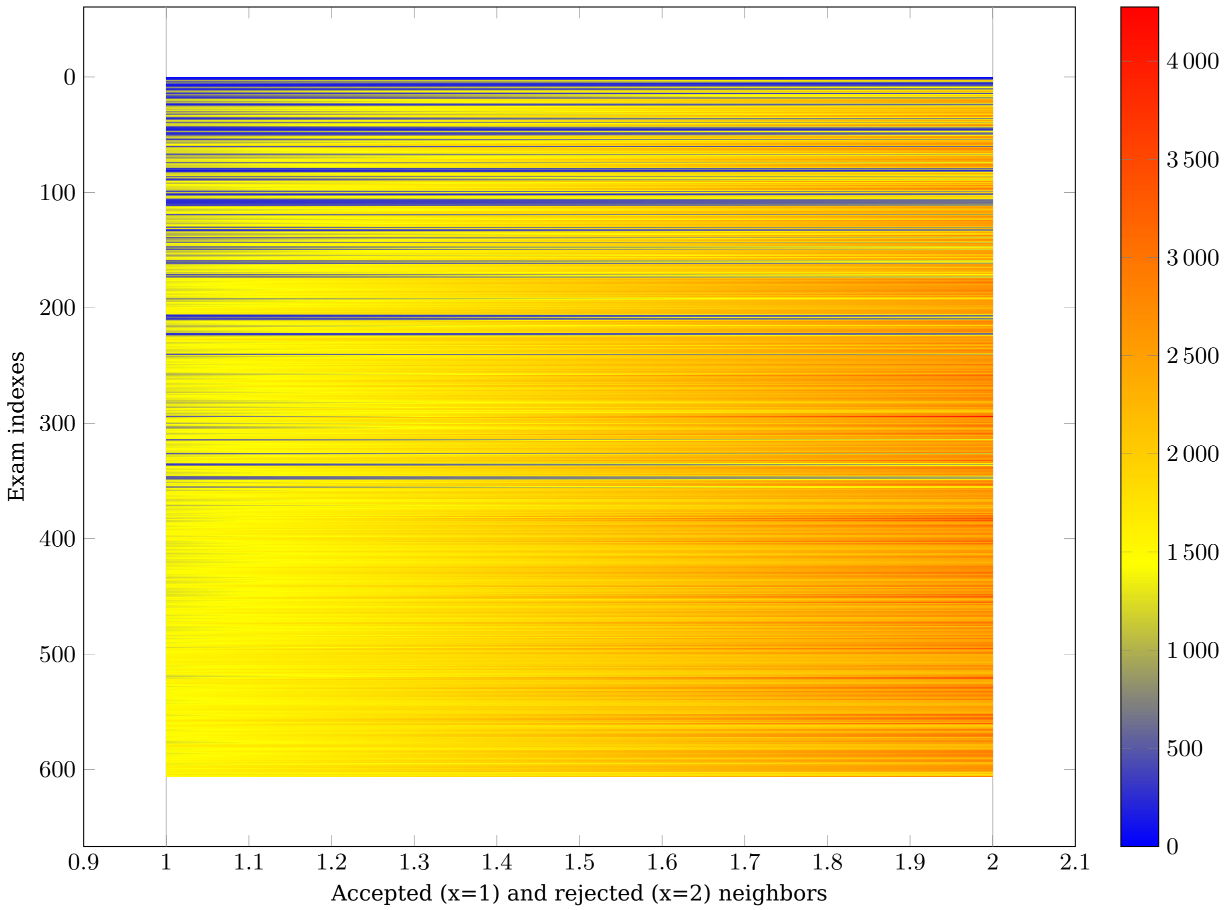

Figure 5.5: Accepted and rejected neighbors for each examination in the Simulated Annealing.

5.3.3 Neighborhood Operators Effect

One of the most important aspects when dealing with the SA is the choice of the neighbor operators. It is crucial that all the neighbor operators contribute positively. If a neighbor operator contributes negatively, it's better not to use it. Müller proposed the first five operators [14] *Room Change*, *Period Change*, *Period & Room Change*, *Room Swap*, and *Period Swap*. We add to the Müller's set a sixth operator, the *Period & Room Swap* operator. In the sequel we present a study of the influence of this operator. For all 12 datasets, 20 runs were performed, computing the fitness average for each of the two cases. Table 5.1 shows the improvement factor for using six neighbors instead of five. In most cases, it's worse compared to using the original five neighbor operators, and so, the 6th operator will not be used in this project's results. The results of using five operators also showed that more neighbors were generated and the percentage of feasible generated neighbors compared to the unfeasible, for each dataset, is also higher compared to using six neighbor operators.

Table 5.1: Improvement factor (percentage) between using five and six neighbor operators.

Dataset	5 Neighbors Operators	6 Neighbor Operators	Improvement Factor (%)
1	5019	5055	-0.7
2	478	523	-9.4
3	14 997	15 228	-1.5
4	–	–	–
5	4950	4819	+1.5
6	27 680	27 528	+0.5
7	4799	4812	-0.3
8	8551	8741	-2.2
9	1141	1150	-0.8
10	28 219	25 586	+9.3
11	45 344	45 630	-0.6
12	7222	7679	-6.3
$\sum$	148 400	146 751	-10.5

5.4 Fitness Computation

Fitness evaluation in timetable automation, due to its complexity, has a great influence in the algorithm performance. Hence, good performance of this step is required. As mentioned in the Subsection 3.4, this value is computed using the `EvaluationFunction` tool, which in a first approach, implies for generated neighbors, the recalculation of their fitness value. Tests were made to study this approach since the fitness results were not as good as expected. The tests showed that this approach was having poor performance, since most of the time was used to compute the fitness for each new neighbor generated solution.

A new approach was implemented, which consists in computing the fitness incrementally. This means that for each new generated neighbor, a new fitness value is computed based on the current solution's fitness. This approach takes advantage of the fact that the operations applied to the current solution are simple and so the neighbor solutions are, in most part, equal to the current solution. Thus, the fitness computation is performed considering only the small changes between the current solution and the new generated neighbor solution.

Tests were made to compare both approaches and, using the same parameters, for the first dataset it was possible to achieve equivalent results, 19 times faster. As a consequence if both approaches would use both the same parameters, and execution time, the new approach would have generated 19 times more neighbors, thus exploring the solution space with more detail.

5.4.1 Implementation

In order to compute the fitness of the neighbor solutions incrementally, a fitness function must be implemented depending on the type of neighbor. Considering that all neighbors have a reference to the solution they derive from, and the methods `Accept` and `Reject`, it's possible to access the current solution and the neighbor generated solution (`Accept` will change the current solution and so the reference presented in this neighbor, replacing with the neighbor solution).

The differences between the fitness values of the current solution and the new neighbor's depend on the type of neighbor created. For example, if a `RoomSwapNeighbor` is created, the changes will occur in the mixed durations constraints value of the periods and rooms of the swapped examinations. It is necessary to compute the impact (fitness values) of these conflicts in the current solution and subtract their values to the current fitness, proceeding with computing the same impact but now for the generated neighbor and to sum these values to the current fitness, thus obtaining the new neighbor's fitness value. The impact computing, mentioned above, is a simple rule that must be followed by all the soft constraints when computing the new neighbor's fitness incrementally.

6

Experimental Results

In this chapter we report the results obtained by the heuristics explained in the previous chapters. We also compare our results against the ones of the top five winners of the ITC 2007 contest, as well as with those of more up to date approaches.

The SA parameters are:

- TMax = 0.1 $\rightarrow$ Maximum/initial temperature;
- TMin = 1e-06 $\rightarrow$ Minimum temperature;
- reps = 5 $\rightarrow$ Number of repetitions per temperature;
- rate = *computed automatically using Algorithm4* $\rightarrow$ Temperature cooling rate.

All the 12 datasets were tested by running the algorithm on each dataset 20 times. Table 6.1 presents the obtained results, showing the values for the average fitness, fitness' best and worst values, its standard deviation, and the average of feasible and unfeasible generated neighbors. Compared to the previous approaches, which excludes the use of incremental fitness computing and the real time computed rate, these results are noticeable better, as can be seen in Table 6.2.

While analyzing the results, it was noticed that some runs produced results very far from the average results for the given dataset. As can be seen in Table 6.1, the datasets 2, 5, 6, 8, 10, and 12 have the worst fitness deviating from the average fitness compared to the other datasets. The fitness standard deviation itself, in these datasets, is much smaller compared to the distance between the average fitness and the worst fitness of the 20 runs. On the 20 runs for these datasets, only one or two had this kind of results. We believe this rarely happens because of the problem specified in Subsection 5.3.2, the fact that examinations with coincidence hard constraints cannot be moved, it limits the algorithm performance to get better results.

In Table 6.3 we compare the results obtained in this project's approach with the five winners of the ITC 2007. Our approach is based on Müller's [14] approach, however the results are not as good as his, and this might be because of the use of the GD heuristic in his approach and differences on the neighborhood and on the initialization procedure. In some datasets,

Table 6.1: Obtained results for the proposed SA hybrid algorithm. "–" indicates that a feasible solution could not be obtained.

Dataset	Average Fitness	Best Fitness	Worst Fitness	Fitness Standard Deviation	# Feasible Generated Neighbors	# Unfeasible Generated Neighbors
1	5019	4857	5307	123	2 004 135	12 937 397
2	478	451	921	102	3 220 534	4 953 314
3	14 997	13 375	17 755	1121	663 246	4 169 914
4	–	–	–	–	–	–
5	4950	3965	7737	1003	2 273 723	8 970 543
6	27 680	26 665	30 145	828	2 105 394	14 922 008
7	4799	4576	5448	282	1 732 856	3 423 649
8	8551	8238	11 670	687	1 889 858	6 006 218
9	1141	1033	1300	70	4 610 766	18 751 578
10	28 219	19 858	38 698	4778	768 633	9 663 356
11	45 344	39 150	53 611	4113	561 047	4 888 157
12	7222	6109	12 033	1684	2 287 229	24 911 726

Table 6.2: Comparison of the fitness results of the current approach with the previous approach. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained.

Dataset	Previous approach	Current approach	Improvement Factor (%)
1	6934	**5019**	+27.6
2	821	**478**	+41.8
3	24 627	**14 997**	+39.1
4	–	–	–
5	7729	**4950**	+36.0
6	30 195	**27 680**	+8.3
7	8089	**4799**	+40.7
8	11 067	**8551**	+22.7
9	1448	**1141**	+21.2
10	35 698	**28 219**	+21.0
11	72 751	**45 344**	+37.7
12	8049	**7222**	+10.3
$\sum$	207 408	148 400	+306.4

however, the developed approach was able to reach and beat most of the contestants. We can also observe that the datasets with the highest scores, and so represent the ones with worst results, are the ones that have more *exam_coincident* hard constraints. We believe that by solving this approach limitation mentioned above, would improve the algorithm results.

In Table 6.4, our algorithm's results are compared to more up to date approaches. As can be seen, Rahman et al. [39], Bryce et al. [38], and Mouhoub et al. [37] beat almost all the

Table 6.3: Comparison of the proposed approach with the ITC 2007 finalists. The comparison is made between the average values of each approach. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained.

Dataset	Müller 2009 [14]	Gogos et al. 2012 [19]	Atsuta et al. 2007 [20]	De Smet 2008 [4]	Pillay 2008 [22]	Proposed Approach 2015
1	**4370**	5905	8006	6670	12 035	5019
2	**400**	1008	3470	623	3074	478
3	**10 049**	13 862	18 622	–	15 917	14 997
4	**18 141**	18 674	22 559	–	23 582	–
5	**2988**	4139	4714	3847	6860	4950
6	**26 950**	27 640	29 155	27 815	32 250	27 680
7	**4213**	6683	10 473	5420	17 666	4799
8	**7861**	10 521	14 317	–	16 184	8551
9	**1047**	1159	1737	1288	2055	1141
10	16 682	–	15 085	**14 778**	17 724	28 219
11	**34 129**	43 888	–	–	40 535	45 344
12	5535	–	**5264**	–	6310	7222

Table 6.4: Comparison of the proposed approach with state-of-the-art approaches. The comparison is made between the average values of each approach. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained, or the following datasets were not tested.

Dataset	Rahman et al. 2014 [40]	Bryce et al. 2014 [38]	Mouhoub et al. 2014 [37]	De Smet 2014 [36]	Abdullah et al. 2014 [35]	Proposed Approach 2015
1	5231	5302	**4395**	6235	5517	5019
2	433	**418**	433	2974	537	478
3	**9265**	10 036	10 118	15 832	10 324	14 997
4	17 787	20 531	21 772	35 106	**16 589**	–
5	3083	3236	**2836**	4873	3631	4950
6	**26 060**	26 253	27 166	31 756	26 275	27 680
7	10 712	**4115**	4294	11 562	4592	4799
8	12 713	**7555**	7632	20 994	8328	8551
9	1111	**1089**	1109	–	–	1141
10	**14 825**	15 167	17 022	–	–	28 219
11	**28 891**	31 415	33 608	–	–	45 344
12	6181	**5464**	6364	–	–	7222

best results. Some of Rahman's [39] and Bryce's results surpass Müller's [14] results. None of the datasets' score of our approach was able to beat all the other approaches shown on the table.

One interesting test would be to run the created approach allowing it to run for a large period of time, not considering the time limit. The SA heuristic should obtain better results compared to the 221 seconds time limit tests. Table 6.5 reports the comparison of the origi-

Table 6.5: Results with different time constraints. The best solutions are in boldface. "–" indicates that a feasible solution could not be obtained.

Dataset	221 seconds fitness	12 hours fitness
1	5019	**3784**
2	478	**385**
3	14 997	**12 928**
4	–	–
5	4950	**3681**
6	27 680	**25 890**
7	4799	**3260**
8	8551	**7011**
9	1141	**978**
10	28 219	**19 734**
11	45 344	**31 366**
12	7222	**5580**

nal tests with time limit, with tests that took 12 hours for each dataset. It can be seen that for all the datasets, the 12 hours tests, the algorithm obtained better results.

7

Conclusions

In this project, an algorithm to solve the timetabling problem, was proposed. It was the main objective to use the problem described in the ITC 2007, and so the instances presented in this competition were used. The developed algorithm utilizes an hybridization of heuristics (GC) and meta-heuristics (SA and HC), to solve the problem.

With the use of this hybrid approach of GC, SA, and HC we've reached near the results of the five finalists of the ITC 2007 on most of the datasets. In some datasets, we were able to be right before Müller's results, the winner of the competition.

Our results were compared against the top five contestants of the competition, as well as with state-of-the-art approaches. These latter approaches were able to beat Müller's results on most of the datasets, but Müller's approach can still get very good results on all the datasets. Müller's results contrast with some approaches that can beat his results on some datasets but get much worse results on others.

We can conclude that the SA works as predicted, getting much better results on most of the datasets when executed for much more time. Running the current approach for 12 hours, yielded better results, as compared to Müller's and some recent approaches.

7.1 Future Work

There are different approaches to be tested with the implemented approach in order to improve the results. Some new approaches may include the use of more heuristics with the SA and HC, such as GD. One improvement that must be implemented in order to get better results is editing the implemented SA and make it able to move more than one examination at each time, to allow examinations with *exam_coincidence* hard constraint to be moved. It is expected that after implementing this improvement, the number of rejected moves might get lower, leading to produce better results.

An extensive study should take place, in order to try and find feasible solutions for the dataset 4. To reach this goal, different approaches should be implemented and tested using the GC.

References

1. E. Talbi, *Metaheuristics - From Design to Implementation*. Wiley, 2009.
2. 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, pp. 146 – 168, May 2012. Data Mining for Software Trustworthiness.
3. S. Kirkpatrick, D. Gelatt, and M. Vecchi, "Optimization by simmulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
4. G. Smet, "Drools-solver." `http://www.cs.qub.ac.uk/itc2007/winner/bestexamsolutions/Geoffrey_De_smet_examination_description.pdf`, 2007.
5. A. Schaerf, "A survey of automated timetabling," *Artif. Intell. Rev.*, vol. 13, no. 2, pp. 87–127, 1999.
6. T. Jensen, *Graph Coloring Problems*. John Wiley & Sons, Inc., 2001.
7. S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
8. 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.
9. R. Lewis, "A survey of metaheuristic-based techniques for university timetabling problems," *OR Spectrum Volume 30, Issue 1 , pp 167-190*, 2007.
10. P. Boizumault, Y. Delon, and L. Peridy, "Constraint logic programming for examination timetabling," *The Journal of Logic Programming*, vol. 26, no. 2, pp. 217 – 233, 1996.
11. S. Al-Yakoob, H. Sherali, and M. Al-Jazzaf, "A mixed-integer mathematical modeling approach to exam timetabling," *Computational Management Science*, vol. 7, no. 1, pp. 19–46, 2010.
12. M. Carter, G. Laporte, and S. Lee, "Examination timetabling: Algorithmic strategies and applications," *The Journal of the Operational Research Society*, vol. 47, pp. 373 – 383, Mar. 1996. 1996.
13. B. McCollum, "ITC2007 examination evaluation function." `http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/examevaluation.htm`, 2007.
14. T. Müller, "ITC2007 solver description: A hybrid approach," *Annals of Operations Research*, vol. 172, no. 1, pp. 429–446, 2009.
15. T. Müller, *Constraint-Based Timetabling*. PhD thesis, Charles University in Prague Faculty of Mathematics and Physics, 2005.
16. T. Müller, R. Barták, and H. Rudová, "Conflict-based statistics," tech. rep., Faculty of Mathematics and Physics, Charles University Malostranské nám. 2/25, Prague, Czech Republic, 2004.
17. S. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.
18. G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *Journal of Computational Physics*, vol. 104, pp. 86 – 92, January 1993.
19. 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.

20. M. Atsuta, K. Nonobe, and T. Ibaraki, “ITC-2007 track2: An approach using general csp solver,” tech. rep., Kwansei-Gakuin University, School of Science and Technology, Tokyo, Japan, 2007.
21. J. Drools, “Drools planner user guide.” `http://docs.jboss.org/drools/release/5.4.0.Final/drools-planner-docs/html_single/`.
22. N. Pillay, “A developmental approach to the examination timetabling problem.” `http://www.cs.qub.ac.uk/itc2007/winner/bestexamsolutions/pillay.pdf`, 2007.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. H. Turabieh and S. Abdullah, “An integrated hybrid approach to the examination timetabling problem,” *Omega*, vol. 39, no. 6, pp. 598 – 607, 2011.
29. 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.
30. E. Burke and Y. Bykov, “A late acceptance strategy in hill-climbing for examination timetabling problems,” in *Proceedings of the conference on the Practice and Theory of Automated Timetabling*, 2008.
31. 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.
32. N. Sabar, M. Ayob, R. Qu, and G. Kendall, “A graph coloring constructive hyper-heuristic for examination timetabling problems,” *Applied Intelligence*, vol. 37, no. 1, pp. 1–11, 2012.
33. 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.
34. S. Abdullah and M. Alzaqebah, “A hybrid self-ddaptive bees algorithm for examination timetabling problems,” *Applied Soft Computinglr*, vol. 13, no. 8, pp. 3608–3620, 2013.
35. 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.
36. 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.
37. A. Hmer and M. Mouhoub, “A multi-phase hybrid metaheuristics approach for the exam timetabling,” *Department of Computer Science University of Regina Regina, Canada*, 2014.

38. R. Hamilton-Bryce, P. McMullan, and B. McCollum, "Directed selection using reinforcement learning for the examination timetabling problem," *10th International Conference of the Practice and Theory of Automated Timetabling*, pp. 218 – 232, August 2014.
39. 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.
40. 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.
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. B. McCollum, "ITC2007 examination input format." `http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/Inputformat.htm`, 2007.
44. B. McCollum, "ITC2007 examination benchmarking." `http://www.cs.qub.ac.uk/itc2007/index_files/benchmarking.htm`, 2007.
45. P. Carvalho, "Lecture notes in evolutionary computation." Instituto Superior Técnico, November 2004.