INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

GuideMe - Service for Consulting, Publication and Recommendation of Touristic Locations

ARTEM UMANETS

(Licenciado em Engenharia Informática e de Computadores)

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

Orientadores:

Mestre Artur Jorge Ferreira

Mestre Nuno Miguel da Costa de Sousa Leite

Novembro de 2014

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

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

GuideMe - Service for Consulting, Publication and Recommendation of Touristic Locations

ARTEM UMANETS
(Licenciado em Engenharia Informática e de Computadores)

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

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

Novembro de 2014

Acknowledgments

Resumo

Palavras Chave

Abstract

Keywords

Contents

List of Figures

List of Tables

List of Code Listings

1 Introduction

In this world, most of the people think that Artificial Intelligence was created to simulate human behavior and the way we humans think. Even thought people are not wrong, Artificial Intelligence was also created to solve problems that humans are unable to solve, or to solve it in a shorter amount of time, with a better solution. For example, any exhaustive search requires pretty complex algorithms (heuristics, meta-heuristics etc) with complex mathematical functions in order to find a feasible solution. Problems humans may take days to find a feasible solution, or may not find a solution at all that fits its needs, algorithms like these 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 better solution and not just a feasible one.

A concrete example is the creation of timetables for scheduling. Timetables can be used for educational purposes, sports scheduling, transportation timetabling etc. It may look like it's simple to create a timetable, but the truth is, this process normally requires following some rules in order to fit its purpose. These rules are called constraints in the timetabling subject, in which the more it has the harder it gets to create a feasible solution for the problem. In some cases, there are so many constraints in which some can be contradictory to each other, that it is needed to "relax the constraints" in order to solve the problem.

The process of creating a timetable requires that the final solution follows a set of constraints. These can be divided in two groups: hard constraints and soft constraints. Hard constraints are a set of rules which must be followed in order to get a feasible solution. In the other hand, soft constraints may be called "optional" since there is no need to follow these to get a solution, because they are not mandatory. The quality of a solution is calculated by using these soft constraints. In another words: the more soft constraints a solution follows, the better it is. This quality is based on the points (weight) of the soft constraints that that solution didn't follow, so the more points, the worse the solution. The weight of each soft constraint is set by the one that created the constraints.

Now that I talked a little bit about timetabling, let me specify the aim of this project. Its main is to create an examination timetable generator using ITC-2007 specifications. The solutions will be validated using a validator also created by me which specifies the quality of the solution and may do some corrections on the timetable in order to get an even better solution. It is also required that the final product may work with two test seasons which can be considered an extension to ITC-2007 formulation. In the end an (optional) Graphical User Interface will be created in order to allow the user to edit the current solution to fit the users needs and allow optimization to the edited solution. The generator will be tested using data from ITC-2007 and some actual data from six different programs presented in my university ISEL.

1.1 Examination timetable: State of the Art

In this topic we'll be writing about the state of art concerning examination timetable. Concretely, we'll be writing about why timetabling is a rather complex problem, some possible approaches on trying to solve a problem this type and some of the solutions already taken, specifically in ITC 2007.

1.1.1 Timetabling Problem

Timetabling is a subject that has been under investigation for more than ten years. There is no optimal solution for creating perfect timetables. Creating timetables is a process that requires complex algorithms in which search for solutions following a set of constraints, as mentioned above. These constraints can be divided into five main classes named Unary, Binary, Capacity, Event Spread and Agent constraints (REF: 2008 Rhydian Lewis)

Timetabling problem may be formulated as a search or optimization problem (REF: 1999 Schaerf). Search problems consists about finding a solution that satisfies all the hard constraints, which soft constraints aren't the main goal, on contrary optimization problems are problems which tries to satisfy as most soft constraints as possible after satisfying all hard constraints. Optimization problems though are more commonly used to optimize feasible solutions obtained by using a search algorithm. The main goal consists in searching for a feasible solution which satisfies all hard constraints using a search algorithm. If the user's goal is to find the best solution possible in given time, a optimization solution may be applied to the solution given by the search algorithm.

Search problems are quoted as NP-Complete (REF: NP-Complete wiki) and optimization problems are quoted as NP-Hard (REF: NP-Hard wiki) problem. Optimization problems are labeled NP-Hard because any optimization problem can be reduced to a graph coloring problem which is also NP-Hard. Graph coloring will be briefly explained later.

1.1.2 Solution approaches

Timetabling solution approaches may be the most difficult decision making part of the project. Considering the number of possible solution approaches that can be made, but still only a short number of those approaches may have good solutions within a reasonable amount of time. Considering the state of art, some combinations of algorithms appear to be better than others, if well implemented and optimized. These algorithms normally are divided in search algorithms and optimization algorithms in which are known as Heuristics and Meta-heuristics. Both of these can be used to generate solutions separately, but normally Heuristics are used to generate a solution not guaranteed to be optimal, but good enough to at least solve all the hard constraints. In contrary the Meta-heuristics are often only to, given a feasible solution, generate an even better optimized solution, in which the main goal is to solve the most soft constraints possible in order to get the lowest score. Heuristics though are problem-dependent, meaning that these are adapted to a specific problem in which take advantage of its details. Considering these algorithms are often greedy, they tend to get trapped on local optimum. Meta-heuristics are problem-independent, meaning they can't take advantages of specific problems. These are often not greedy so they can be used to solve (optimize) any problem given.

Most of the Meta-heuristic algorithms used belong to one of the three categories: One-Stage algorithms, Two-Stage algorithms and Algorithms that allow relaxations. (REF: 2008 - Rhydian Lewis). The first two algorithms are worth explaining, mainly because of their usage in the state of art. The One-Stage algorithm works as Meta-heuristic algorithm(s) that is only used to get an initial optimal solution, which the goal is to satisfy both hard and soft constraints at the same time. Approaches using this are not very common because it's hard to get proper solutions in a reasonable amount of time trying to satisfy both types of constraints at the same time and this one of the reasons the Meta-heuristics are only used as optimization algorithms instead of search algorithms. The Two-Stage algorithms are the most used types of approaches because two phases (the reason for "Two-Stage" name) which the first phase consists in all soft constraints being "discarded" and focus only on solving hard constraints to obtain feasible solution. The next phase is an attempt to find the best solution, trying to solve the highest number soft constraints possible.

Some approaches may use methods dominated *exact methods* which may be viewed as tree algorithms and can be proved that it can find the optimal solution (global optimal). Exact methods search for solutions in the whole search space, and it divides the global problem into simpler problems in order to find the solution. This is not always used because these methods require a lot of computing time and it rarely produces results in a reasonable time in complex problems. For example, if the problem is complex enough it may require dozens of processors to obtain the best solution in a month of computing time. Approaches use some of these exact methods, named: Constraint Programming Based Technique, Integer Linear Programming.

Constraint Programming Based Technique:

This type of technique allows direct programming with constraints which gives ease and flexibility in solving problems like timetabling. Two important features about this technique that are used are backtracking and local variables that facilitates searching for an optimal solution (with a great cost: time). Constraint programming is different from other types of programming, as in these types it is specified the steps that need to be executed, but in constraint programming it is specified the properties (hard constraints) of the solution or properties that should not be in the solution. (REF: 2009 - Qu Burke)

Integer Linear Programming:

Integer Linear Programming is a technique in which some or all variables must be integer and the objective function and the constraints (non-integer) must be linear. Schaerf in his article specifies some approaches to school, course and examination timetabling using all integer linear programming methods to obtain feasible solutions and perform optimization (REF: 1999 - Schaerf).

The usual approaches start with using heuristics of Graph Coloring to get an initial solution that most of the cases is a local optimum. Graph Coloring itself is not an heuristic or meta-heuristic but a method that designates a problem and its variants:

Graph Coloring:

This algorithm is divided in two main sub-types, which is vertex coloring and edge coloring. The main goal of this algorithm (vertex) is to, given a number of vertices and edges, color the vertexes so that no adjacent vertices have the same color. In this algorithm, it's best to find a solution with least colors possible. In examination timetable problem, a basic approach could be to represent the exams as vertices and the hard constraints as edges (considering this is search algorithm, it is good to use optimization algorithms to deal with soft constraints) so that exams with the same color, can be assign to the same timeslot. After coloring, it proceeds to assign the exams into timeslots considering the colors of the solution. (REF: 2009 - Qu Burke)

Graphic Colors heuristics like Saturation Degree Ordering are very commonly used to get the initial solutions. Others like First Fit, Degree Based Ordering, Largest Degree Ordering, Incident Degree Ordering are also heuristic techniques for coloring graphs.

Saturation Degree Ordering:

This heuristic is very useful because it colors the vertices with more constraints first. The coloring method is as follows: while choosing a vertice to color, the ones with higher saturation degree will be colored first. Saturation degree of one vertice can be denominated as the number of differently colored vertices adjacent to this vertice or, in another words, the number of different colors of all adjacent vertices. In case of ties, the highest saturation vertice with higher number of adjacent vertices is chosen.

Meta-heuristics on the other hand, as mentioned above, are used to optimize first solutions. On examination timetabling Graph Coloring is used to get feasible solutions, and after that, Meta-heuristics are used to get even better solutions, optimizing the ones obtained via Graph Coloring. Some Meta-heuristics used are named *Evolutionary Algorithms* [REF: Glover and Kochenberger 2003; Reeves 1993; Sastry et al. 2005], *Ant Colony Optimization*[REF: (Dorigo and Blum 2005; Merkle and Middendorf 2005], *Tabu Search*[REF: Gendreau and Potvin 2005; Glover and Laguna 1993], *Simulated Annealing* [REF: Aarts and Korst 1989]. For more details about these Meta-heuristics (REF: 2009 - Qu Burke).

For the common types of algorithms and how they are organized, please check Figure 1.1

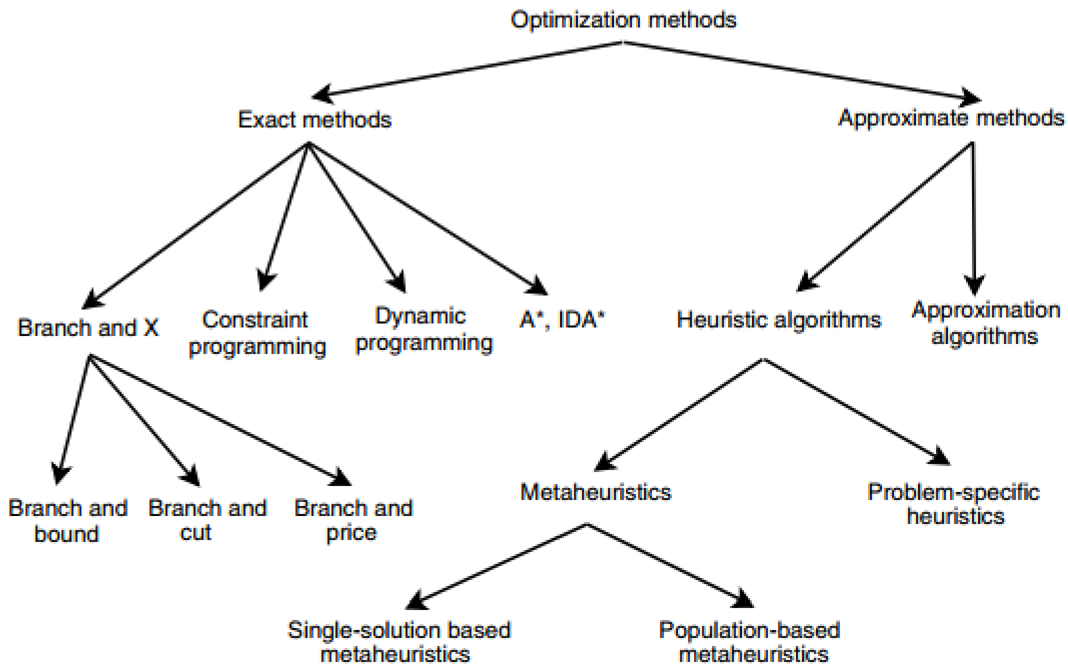

Figure 1.1: Types of algorithms.

1.1.2.1 Some approaches applied to the ITC 2007 Examination timetabling problem

I will briefly describe some techniques used in ITC 2007 - Examination timetabling, as detailed explanations about its techniques are properly presented in its articles which will be referenced below. ITC 2007 Examination timetabling problem consists on submitting a solution which may solve all hard constraints and the quality of each solution is calculated using the soft constraints the the solution didn't solve. Each soft constraint has its weight in the problem's context, and that weight adds points to each other solution. In this case, the more points a solution has, the worse it is. Solutions with zero points are considered optimal solutions, because they were able to solve all hard and soft constraints. Each of these constraints are presented on ITC's site (REF: ITC's site) or it can be read in Tomas Muller's article as he explains each constraints, including for other educational timetabling problems, named *post enrollment based course timetabling*, and *curriculum based course timetabling*

Tomas Muller's approach:

Tomas Muller's approach was actually used to solve all three problems established by ITC 2007 competition. He was able to win two of them and be finalist on the third. For solving the problems, he opted for an hybrid approach. Meaning it has more than one algorithm working at the same time for first (construction) phase and/or for the second (optimization) phase. In this case, he chose to use multiple algorithms to optimize its first solutions.

In the first phase, Tomas used Iterative Forward Search algorithm (REF: Muller 2005) to obtain feasible solutions and Conflict-based Statistics (Muller et al. 2004) to prevent Iterative Forward Search from looping.

Iterative Forward Search algorithm assigns a value to one unassigned variable on each iteration. These variables are exams (in examination timetabling problem) and are only assignable if the result won't cause violations of hard constraints. If it's the case, some precautions may be applied to continue with the algorithm. An assign will result in an exam (the variable assigned) being held in a room X, time Y with Z students. The variable chosen in each iteration is randomized or parameterized to, for example, assign the most difficult assignable exam first. The Conflict-based Statistics was used to memorize some previously passed conflicts and avoid repeating those for each iteration.

The second phase consists in using multiple optimization algorithms. These algorithms are applied using this order: Hill Climbing, edited Great Deluge (REF: Dueck 1993) and optionally Simulated Annealing (REF: Kirkpatrick et al. 1983).

Hill Climbing is used to optimize the current solution only, by generating a neighbor solution each iteration. The neighbor solution is only accepted if the "score" of this solution is lower than the one we had before (only soft constraints count now) and no hard constraint is violated. If there is no better solutions for a number of iterations, which is parameterized, this algorithms stops running and the Great Deluge starts doing its job.

Great Deluge, used after the solution cannot be improved using the previous algorithm. This method can be seen as a Hill Climbing variation, as it always accepts going up but is also able to go down the Hill as long as it doesn't go lower the water level. This means that it accepts worse neighbor solutions compared to the current solution, but only accepts worse solutions which absolute value is less compared to the value of the Bound, as the Bound is the variable that corresponds to the water level mentioned earlier. The water level is always going up, until reaching a local/global maximum. The initial Bound and the way the Bound decreases (water level going up) is parameterized.

When Simulated Annealing (SA) is used (remember that this method is optional), it is used after Great Deluge reaches his bound lower limit. If this method is not meant to be used, Great Deluge might be running until reaches timeout or a global maximum. SA is a method similar to Hill Climbing, as it accepts neighbor solutions better than the current one. But this method also allows neighbor solutions that are worse than the current one, by establishing a mathematical operation using a variable called Temperature. The result of this operation stands for "how much wrong can the solution become" in order to avoid local maximums.

This variable called Temperature is parameterized so as the Cooling Rate variable which is multiplied by the Temperature each iteration, in order to reduce the Temperature. This means that for each iteration, neighbor solutions will get less worse and it is possible to reach a local maximum again. If the local maximum is the same (solution is the same as before running Simulated Annealing), that means the maximum Temperature must get higher in order to avoid the local maximum and/or to possibly get a better solution. As the Temperature gets higher, Simulated Annealing is not repeated, but instead, Hill Climbing takes control again.

This is the basics and a resume of Muller's approach. More detailed explanations and implementations can be viewed on this article (REF: 2009 - Thomas Muller - ITC2007 solver description a hybrid approach).

Christos Gogos' approach:

Gogos was able to reach second place in Examination Timetabling, right after Muller. Gogos' approach is very different compared to Muller's. His approach, like Muller's, was divided in 2 phases named *Construction Phase* and *Improvement Phase*, which the first is to get a feasible solution and the second is to improve the solution found in the first phase, using local search algorithms.

In the first phase, is starts using a Pre-processing stage. It is called *Pre-processing* because it's a stage that adds additional information to the main problem in order to make it easier to handle in the further steps. This stage deals with hidden dependencies, which means it adds dependencies to the problem that weren't there in the beginning but they exist and makes sense in the problem itself. For example, one hidden dependency may be one exam E1 having an ordering constraint with exam E2 stating that E2 must be scheduled after exam E1 and exam E3 must be scheduled after exam E2. So a dependency/constraint must be added so exam E3 must be scheduled after exam E1 as well. Gogos believes that this pre-process method will be very helpful in the further stages when trying to find solutions.

After the pre-processing stage, a construction stage takes place. This method is used to create a complete timetable. In this stage, multiple solutions are attempted considering the available time in which only the best solution is passed to the next phase. The solution is constructed by constantly setting an exam and a room to a period until a feasible solution is found. This selection is not random, instead some rules must be applied. This method however is not a very known or common algorithm.

In the second phase, Hill Climbing is used. This algorithm is used to get generate neighbor solutions, accepting only better solutions compared to the current one, until reaching a local maximum. It is considered local maximum when Hill Climbing cannot generate a better solution in a number amount of tries or time limit.

Next stage consists in utilizing Simulated Annealing to get out of local maximum. Simulated Annealing skipped when it cannot generate better solutions after a set period of time.

After Simulated Annealing, an Integer Programming formulation that uses Breach and Bound is used in a set of sub-problems. This stage is called "IP Sub-Problems Stage". This simply examines all periods trying to discover some possible improvements that can be done, for example, swapping a room with high penalty with a room with no or low penalty that is not currently being used.

The last stage is the Shaking Stage. This method "shakes" the current best solution in order to get equally good solutions and pass it to Simulated Annealing stage. This method is not used if the problem we're solving only has 1 room or the cost of each room assign is zero. Two heuristics are used in this stage. The first creates neighbor solutions but a solution is only accepted if the new solution is equally good and the room arrangement is better (total cost of room assignment is lower) compared to the current best solution. The second heuristic reschedule a set of exams presented on the latest solution. These are first removed from the timetable and then scheduled using the same techniques used in the construction stage. This heuristic will probably generate a worse solution but it is accepted and used next on Simulated Annealing stage.

Tabu Search is also used in this approach, even tho it's not used as a standalone method. This one is used along with Hill Climbing and Simulated Annealing algorithms to prevent them from looping and creating the same results. Tabu Search avoids the creation of equal neighbor solutions by not letting the same period be selected for swapping after this being selected once. This period can only be selected again after a certain number of swaps on other periods.

This was a brief resume of Gogos' approach. Detailed explanations, implementations, experiments and results are present on this article (REF: 2010 - Gogos et al. - An improved multi-staged algorithmic process for the solution of the examination timetabling problem)

It is possible to check the results of ITC 2007's Examination challenge in Figure 1.2, which illustrates these two top competitors on this challenge. Most of the test instances shows that Muller got the least points (black numbers and dots) and so the reason why he ended up winning. Gogo's results were a little worse (orange numbers and dots) compared to Muller's results and beat the other competitors in most of the instances, not including instances 10 and 12 which not valid solutions were found. And so he got second place on this challenge. All the results can be seen on ITC's original site (REF: ITC's site with results) or on Muller's article (REF: 2009 - Thomas Muller - ITC2007 solver description a hybrid approach)

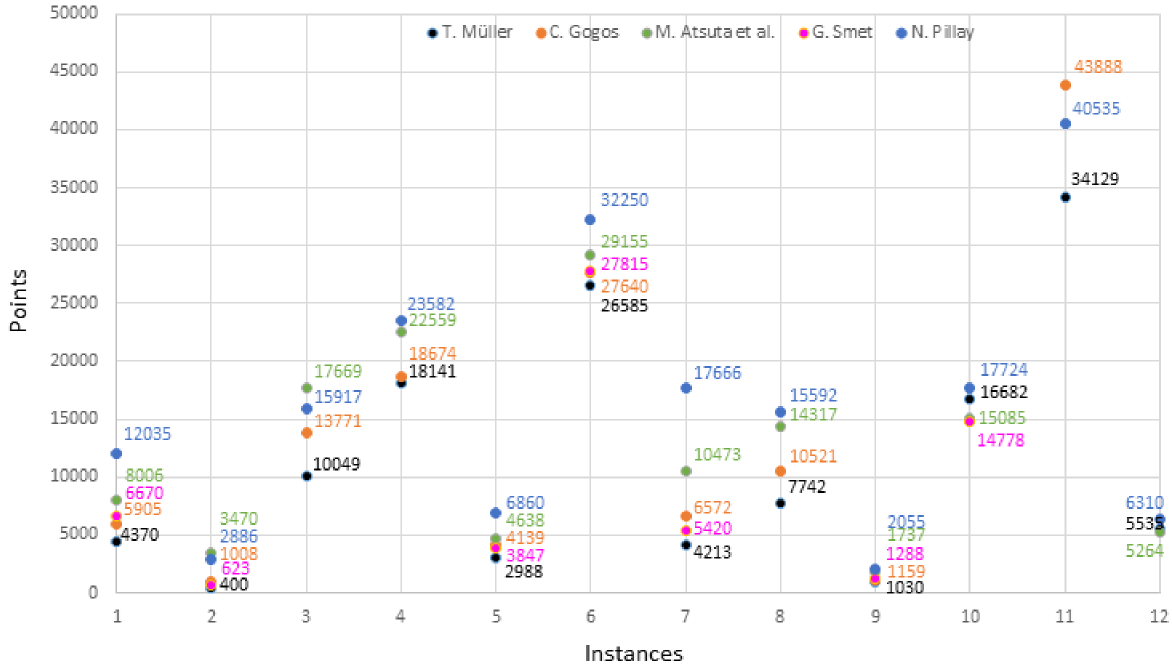

Figure 1.2: Top 5 results on ITC 2007 Examination Challenge