Group 12 - Decision Making and Optimization

Project Report - Examination Timetabling Problem

1 Introduction

This report provides a possible solution approach to the Examination Timetabling Problem by distinguishing between soft and hard constraints. The problem has been structured into two parts: the first part consists in the creation of a feasible solution while the second one requires its optimization by minimizing the objective function, which aims to reduce the penalty due to the proximity of conflicting exams. Two exams are conflicting if they have at least one common student enrolled.

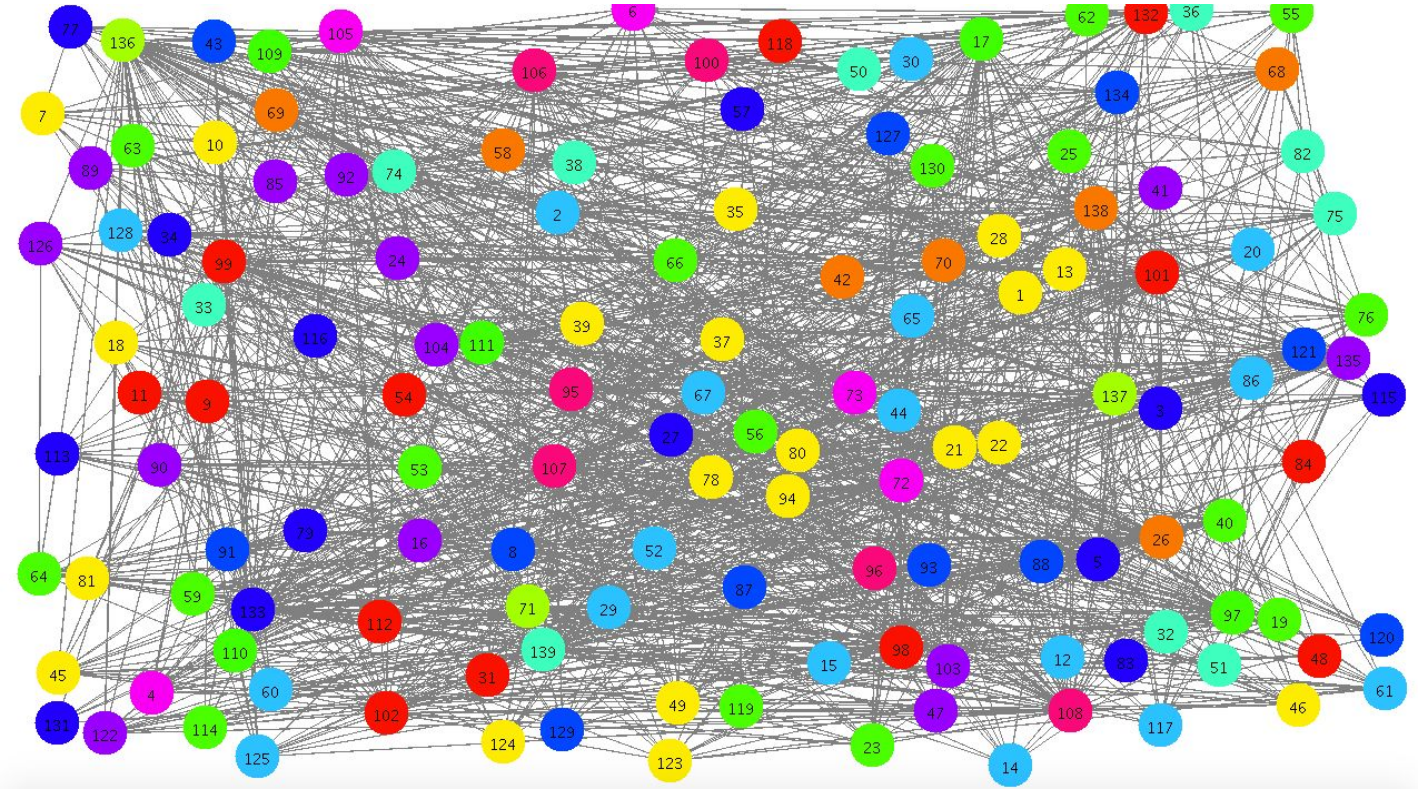
2 Initialization

The first step is to create a population of initial solutions endorsed with feasibility. To serve the purpose, a **greedy Graph Coloring** technique has been adopted. The graph foresees that nodes are represented by the exams while the edges linking nodes represent the conflicts. The coloring has been carried out by using a maximum of t_{max} different colors.

2.1 First feasible solution

In order to find the first feasible solution to insert in the population, the algorithm tries to locate each existing exam in one of its available time-slots. Time-slots get previously organized and ordered depending on a *hash function* so that possible collisions between exams are managed by the *linear probing* technique with a coefficient equal to six; thus, if a time-slot is not available, the exam will be placed six time-slots apart from the current one, if available. Eventually, all time-slots will be investigated for placing the current exam.
The starting ordering of the exams will be executed based on the maximum sum of weighted edges, such that the issues coming from an insertion of those exams at an advanced stage of the algorithm will be reduced. After some iterations, the remaining exams will instead be selected by minimum Saturation Degree, i.e. according to the minimum number of available time-slots for each exam. This way, we reduce the computational time of the algorithm and the number of backtracks needed to end up in a feasible solution.
Nevertheless, it may happen that no time-slot is available for the current exam to place. Under these circumstances, a specific function has been built that allows to shift exams which have been already placed, in such a way to make a time-slot available to locate the current exam. Finally, even if this greedy function is not able to free any time-slot, the algorithm will backtrack to the previous level of recursion and will move the last located exam. In order to prevent an exhaustive graph coloring algorithm (which would be NP), a maximum limit in number of backtracks has then been set, allowing the whole recursion to stop when exceeded.

The following graph shows the first feasible solution we found for Instance01. At first sight is visible how the conflicts graph is very densely connected. Every vertex in the initial feasible solution is colored in such a way that no adjacent vertex have the same color.



2.2 Initial population

The first feasible solution found is inserted in the population. Next solutions will be identified by simply perform random permutations of time-slots of the first feasible solution found (it allows to maintain feasibility), until a population of the requested size has been found.
In some cases, this specific configuration delivers new better solutions with respect to the first one (found in a more deterministic way), in some other cases it does not and the first solution remains the best one of the population.

3 Genetic Algorithm

Once the initial population is obtained, the objective function must be optimized. To do so, the Genetic Algorithm, endorsed with three operators, is executed.

3.1 Rescheduling Operator

The aim of the rescheduling operator is to prevent from getting stuck into local minima; in particular, a random number of exams is first unscheduled and then rescheduled according to the greedy Graph Coloring previously described, to diversify the search and jump over a different region of the solution space. Moreover, worsening solutions will be regardless accepted because they eventually could bring better overall results.
At the beginning, the operator is enabled with a 10% probability that will rise to 40% if it has not been possible to improve the current population after previous multiple generations.

3.2 Swapping Operator

This operator consists in swapping, within the solution, the entire content of two time-slots. Analogously to the approach used to create the initial population, the technique swaps two columns, thus executing a permutation that leaves the feasibility unaltered. The swap occurs between the two time-slots that minimize the objective function while swapped. Therefore, the new generated solution will be accepted only if it brings an improvement with respect to the current 'parent'. This operator will be chosen with the same probability of the third one, in case the first operator is not selected.

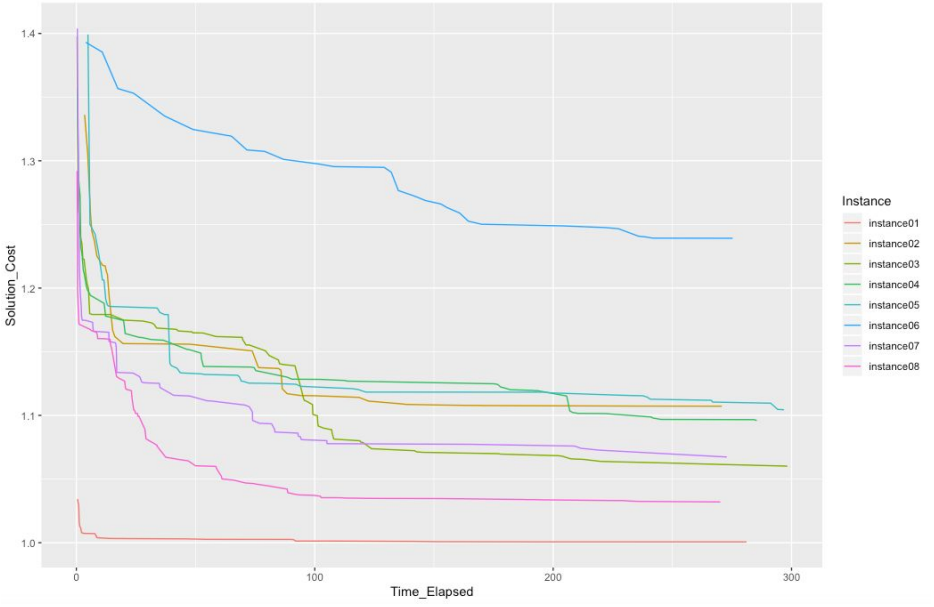
3.3 Multiple Mutation Operator

The latter operator foresees, instead, a classical mutation where, starting from the initial population, the objective is to improve the 'parents' by moving all the exams from their belonging time-slot to another one and obtain a corresponding decrease in the objective function. The exams are ordered based on their contribution to the objective function, i.e. how significant their conflicts are and where they are located. Starting from this ordering, every exam will be relocated to the next available time-slot which minimize the most the objective function. In case the relocation of an exam results in an increase in the objective function, that move will be omitted. As the goal is to ever improve the solution, the algorithm delivers 'children' which are always better than 'parents' with this operator.

As soon as a new generation is created, the population is updated accordingly, and the Genetic Algorithm is performed again. The solutions are ranked according to their goodness, and the first 20% will be saved in the new generation – these solutions will not be modified thereafter in order to preserve the current best individuals. As for the remaining 80%, a random solution is picked and will be subject to improvement. The random pick is executed as many times as the number of the remaining solutions, with replacement from the starting population; therefore, the same 'parent' is allowed to be picked multiple times and this does not harm the algorithm.

4 Results and performances

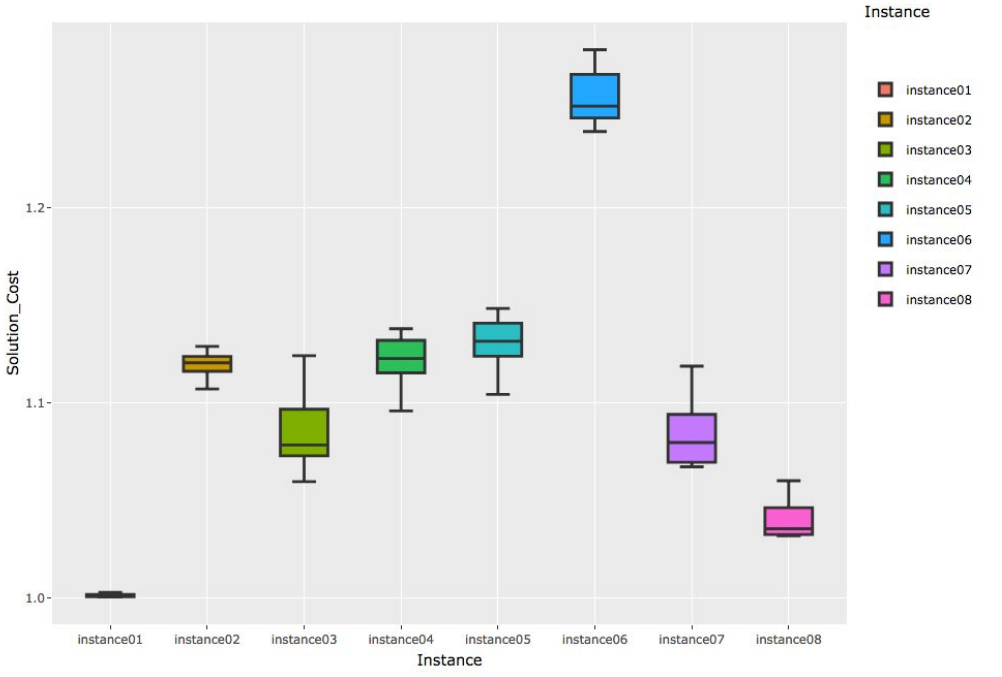
The following chart shows on the y-axis the solution cost reported as ratio with the benchmark, with respect of the elapsed time in seconds, for every instance.



The algorithm outperforms on Instance01 because of the naiveness of the instance. On the other solutions shows similar behaviour because of their ratio of exam with respect to student is very close, with except of Instance06 which have an huge number of student with respect to the number of exams and for which intuitively the penalty would have been higher. It is worth noticing that the algorithm frequently drops after a being

stuck on a local minimum which cause the plot to show a step and fall pattern.

The following chart shows boxplots which represent the variability of each solution calculated in 300 seconds with respect to the instance considered. The variability is induced by our way of choosing randomly the genetic operator to optimize the solution.



Again on Instance01 the algorithm shows low variability because the algorithm outperforms it and the gap with the benchmark is very small at each run. The higher variability is found in those instances (03, 04, 05, 06, 07) which has the highest cardinalities of attributes, this cause the algorithm to significantly varying its final result both because of the limited time and because of the highest likelihood to be stuck more time in a local minima at the increasing of the possible number of configurations.