Optimization methods and algorithms
Assignment - Report

Group 21, course AA-LZ

1 Initialization

This phase consists in the building of an initial feasible solution and it is a metaheuristic as well: it starts from a greedy solution (maybe unfeasible) obtained by graph coloring and then it minimizes the number of conflicts until the solution becomes feasible (that is, there are no exams with common students located in the same timeslot). So the initialization is divided into two main parts: Graph Coloring and a combination of GRASP and Tabu Search.

1.1 Graph Coloring

This part has the following procedure.

1. The exams are ordered (in decreasing order) according to the number of students in conflict with the other exams. This order permits to accelerate the metaheuristic which follows this phase.

2. The algorithm cycles on the exams (following the order arranged in the previous part): at each iteration, the current exam is located in a timeslot in which there are no exams with students in common with the current one. If there is not such a timeslot, the current exam is located in a timeslot chosen randomly.

If this greedy procedure leads to an unfeasible solution, the combination of GRASP and Tabu Search starts, otherwise the initialization phase stops.

1.2 GRASP and Tabu Search

This metaheuristic is carried out by multiple threads: as soon as one of them finds a feasible solution, all of them stops and the algorithm starts with the optimization.

1. Randomly, an exam that still has conflicts with other exams is selected (in order to change timeslot) and the opposite move is put in the Tabu List to avoid to move again this exam in its starting timeslot.

2. The timeslots are ranked according to the number of conflicts which would be generated by moving there the current exam (if the exam is located in the first timeslot, there would be the minimum number of conflicts). Each timeslot has a probability of being chosen (probability of locating the exam in that timeslot) and this probability decreases exponentially thorugh the ranking: the current exam is moved in the selected timeslot.

3. In the selected timeslot, the exams are ranked according to the number of students in conflicts with the other exams in the timeslot: each exam has a probability of being chosen to be potentially moved (the probability decreases as in the previous step). The selected exam is then located in the timeslot in which it generates the minimum number of conflicts (note that it could even not change timeslot).

2 Optimization

In this part of the algorithm different metaheuristics are used.

- **Adaptive large neighbourhood search (ALNS)**
  During the execution we use alternately and flexibly two strategies to generate two different structures of neighbourhood of a solution:
  - SINGLE EXAM MOVEMENT: an exam is moved to a different timeslot (maintaining feasibility).
  - WHOLE TIMESLOT SWAP: all the exams in a timeslot are moved to another one and vice versa. This kind of swapping ensures the feasibility of the neighbours and permits to explore better the solution space: this is due to the fact that if we had only the possibility to move single exams, it could be impossible to find a solution with the same groups of exams, but located in different timeslots, because of possible conflicts during the swapping of all the single exams.
- **Tabu Search (TS)**
  We use a tabu list of variable length to avoid visiting already analysed solutions and to help to escape from local minima.
- **Greedy randomized adaptive search procedure (GRASP)**
  For both the structures of neighbourhood, we rank all the possible moves according to the value of the objective function of the solution they lead to (the first one of this ranking leads to the minimum value of the penalty). Then we assign a probability of being chosen to these moves: the first one in the ranking has the highest probability and this probability decreases exponentially in the ranking. Moreover, we have added some data structures which are very fast to access and update: they permit to reduce the cost of evaluating each neighbour ($\mathcal{O}$(T) for timeslot swaps and $\mathcal{O}$(E) for single exam movements). The GRASP allows to find good solutions in a very short time but sometimes the algorithm gets stuck in local minima and we have adopted some strategies to escape from these situations which are explained below.

The moves from a solution to another one are realized using two functions and each of them has a (dynamic) probability of being used:

1. one is the procedure described in the GRASP;
2. the other one consists in choosing randomly one object (exam or group of exams depending on the current structure of the neighbourhood) to be moved and choosing one move between the best ones involving the selected object.

Moreover, multiple threads are used in the analysis of the neighbourhood.
Before explaining how to understand if the algorithm gets stuck in a local minimum, we remark that at each iteration there are two values of the penalty in the memory: a **total best penalty**, which is the minimum one found until the moment, and a **best penalty**, which is the best one found since the last destruction (see multistart approach below).
There are fixed number of iterations (swapping) per each of the two structures of neighbourhood (e.g. 100 movements of exams, than the algorithm does 50 swapping of groups, and so on). To understand if the algorithm gets stuck in a local minimum we use a counter which starts from 0 and increases each time the algorithm changes the structure of the neighbourhood without having improved the value of the best penalty (if the value improves, the counter is set again to 0).
The more the counter increases, the more likely the algorithm is to get stuck in a local minimum: so the bigger the counter, the higher is the probability of using the function 2 to change the current solution, because that function allows to choose moves which can increase the penalty and so can permit to escape from local minima; in addition, also the length of the tabu list increases, because it is more difficult to come back in a local minimum if more moves are forbidden.
We have also tried different strategies to understand if the algorithm is in a local minimum, such as computing a temperature (simulated annealing) or a sort of distance between different solutions (related to the positions of the exams) or trying to detect the missing of a decreasing trend in the value of the objective function, but none of these seemed to be good.

If the counter reaches the maximum value set in the code (destroy_threshold), the current solution is destroyed and a new one is created from scratch (**MULTISTART APPROACH**). The destruction of the solution is realized by both moving single exams and swapping group of exams, ensuring only feasibility; the tabu list is used in this part too.
We also tried other strategies to destroy the solution, such as using the metaheuristic of the initialization only for some randomly selected timeslots or swapping single exams until the new solution is quite far from the old one (in the sense of the distance mentioned before), but they have proved to be far less effective than the strategy we have chosen.

3 Tuning of parameters

The program we have developed is rich of parameters and so we have wrote special libraries to create logs and scripts to run the program with different parameters each time in order to choose the best ones.
We have collected the results and we have tried some different analyses:

- linear regression to understand the effect of certain parameters on the execution of the algorithm (actually not used, see below);
- graphical analysis of the evolution of the objective function in order to understand if the algorithm got stuck in the same local minima (see Figure 1);
- greedy analysis based on controlling the best solution obtained by varying some parameters.

In these analyses there were two main issues: they needed a lot of time because we had to run the program for a reasonable time (a few minutes) with each set of parameters and so we could not have many executions of the program to analyse (this is why we actually did not use the results of linear regression); in addition, the choice of some parameters depended on the instance of the problem.

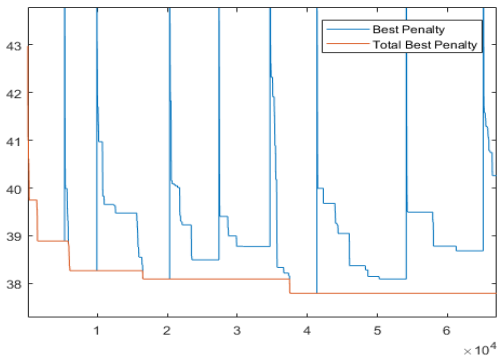

Figure 1: An example of the evolution of the penalties during the execution of the program (on the x-axis there is the number of iterations). Sometimes the best penalty becomes enormous: those are the moments in which a destruction takes place, because in that case we set the best penalty equal to a very big value. It is interesting to note that the solution escapes from local minima and does not reach always the same one, guaranteeing an efficient exploration of the solution space.

4 Results

We have executed the program many times setting different times of execution. The results we obtained running the program on the 7 given instances for 10 seconds were on average around 10% over the benchmarks. Increasing the time of execution, our results reached the 7% in 2 minutes and the 6% in 5 minutes. We also tried to execute the program for more time, reaching less than 5% over the benchmarks.