Examination Timetabling Problem

Group 5

January 2020

1 Main algorithm guidelines

In order to fulfill the assigned task, we decided to follow the subsequent steps:

1. Initialization phase: this produces one feasible solution for the given instance
2. Generation of 11 different neighbors in order to try to improve the current solution
3. Accept or reject one of the new solutions according to Simulated Annealing metaheuristic
4. Repeat 2. and 3. exploiting all available time

2 Finding initial feasible solutions

We will represent the solution as a list with length equal to the number of available time slots: each element consists itself of a list, containing the exams assigned to that time slot. In our algorithm, we first created a list containing all of the exams. Then, we sorted this list with the following criterion: exams with less available time-slots first. The sorting has to be done at each iteration. This type of weighting and ordering is referred in literature as *Least Saturation Degree*[1].

Sorted exams are then assigned to the first available time slot, until all exams are placed. If an exam cannot be placed, the algorithm will exit, and start over from scratch. In the case more exams have the same number of possible time slots, one of them will be chosen randomly. In this way we manage to decrease the amount of times the algorithm run out of available time slots for a given exam.

Moreover, the order of the examined time slots for the assignment of a given exam, is also randomized: the exams division in the timetable is more balanced, hence a much better valuable solution is found.

Note that some decisions in this initialization step are taken randomly, also allowing the algorithm to be more flexible, and deal with situations in which it would otherwise risk to be stuck in infeasibility.

3 Neighborhoods

The different neighborhood structures used in this algorithm are shown in table Table 1.

N1: Swap random exam with random exam	N2: Move random exam to random feasible timeslot	N3: swap all exams of two random timeslots	N4: circulate timeslots
N5/N6: move highest penalty exam from a random 10/20% selection to a random feasible timeslot		N7/N8: move highest penalty exam from a random 10/20% selection to a new feasible timeslot with the lowest penalty costs	
N9: move random exam to a random timeslot and apply the kempe chain algorithm from Thompson and Dowsland (1996a)		N10/N11: same as N9 but move the highest penalty exam from a random 10/20% selection	

Table 1: The 11 implemented neighbourhoods used

4 Simulated annealing and parameters tuning

Neighbourhood constructions provides us with a set of candidate feasible solutions. Better solutions are always accepted: if more than one is available, the best one is chosen. If none of the candidate solutions improves the original solution, the best solution among the worsening ones is accepted according to Simulated Annealing metaheuristic. Solutions are therefore accepted if

$$p < exp\left(-\frac{F(\hat{x}) - F(\bar{x})}{T_k}\right)$$

where:

$F(\hat{x})$ = candidate solution cost function.
$F(\bar{x})$ = current solution cost function.
T_k = temperature parameter at iteration k.
p = a randomly extracted value between (0,1).
Otherwise the current solution does not change.

The temperature, T_k, slowly decreases in the following way:

$$T_k = T_{k-1} \cdot (1 - c)$$

where c is the cooling rate.

This metaheuristic needs the tuning of 2 parameters, which are the initial temperature T_0 and the related cooling rate. We try to tune them following an empirical approach.

We decide to set initial T value as $T_0 = r \cdot t_{lim}$, where t_{lim} is our time limit and $c = s \cdot |E|$ with $|E|$ as the number of exams.

The logic behind this choice is that for larger instances (higher number of exams), the convergence to a stable solution results to be slower with respect to the available time (less iterations for the same time). The chosen value instead, guarantees that in a first time window ranging approximately from 0 to a portion of t_{lim}, worsening solution are accepted quite often. After that, the T value has sufficiently decreased in such a way that the current solution tends to converge around a certain value before the available time elapses.

To sum up: temperature initial value increases with the available time and the cooling rate increases with the size of the instance. Again, the main constraint in this choice reveals to be the available time for running the algorithm.

5 Results

After running the same algorithm for all instances we can see in Figure 1 that, as expected, bigger instances are solved in a fewer number of iterations. However the portion of time in which also worsening solutions are accepted is almost the same for each instance thanks to the parameter tuning, explained in section 4. In general we can say that more or less the quality of the results compared to the given benchmarks is the same for every instance proportionally to the difficulty (see results in appendix Figure 4) if we consider that the gap is normalized on the absolute value of the benchmark cost. Figure 1 also shows how the simulated annealing works, and how the algorithm accepts worsening solutions less and less as the temperature decreases through the iterations.

For what concerns the different neighbours considered in the algorithm we can see from Figure 2 that neighborhoods N2, N5, N6 were not used at all. This may be due to the fact that they are just a worse version of other neighborhoods (N1 and N7/N8 respectively), therefore we decided to eliminate those three neighborhood structures to improve the speed of the algorithm without losing quality of the result.

6 Discussion and conclusion

It is interesting to observe that different neighborhood structures contribute to improving the solution at different time windows: some neighborhoods allow the search to obtain solution improvements in a quick and important manner, but the improvement occurs only for a limited number of iterations (in our case, most of the neighborhood structures). On the contrary, other neighborhoods only enable small improvements, but for a long time (e.g. N1, N9), see Figure 3.

Being the Timetabling problem a very well known problem, some of its possible solution approaches are largely documented. Genetic algorithm are widely used, and in this scope at first we decided to implement our solution using a Memetic algorithm: briefly described, initially generated population is partially mutated by mean of the genetic operator (while crossover operator is not allowed since it cannot guarantee feasibility). Along with it a Local Search is then applied to explore neighbourhoods and find candidate solutions better than the current one. Although promising (see [2] section 9.2 for further reference) this procedure has resulted in being too time consuming for our case and is therefore not suitable.

A way we could have improved our work, would be by doing a more systematic parameter tuning. Since the temperature and the cooling rate are dependent of each other, it would have made sense to do a grid search for the constants, r and s, deciding these. Since our results in average reach values with a 6.3% gap from the benchmark in minutes, and in one instance actually reach the benchmark, we find our algorithm quite efficient and effective.

Please refer to `http://github.com/toyo97/ETPsolver` for further details on the actual implementation of the whole program.

7 Literature

[1] Ayob M., Malik A.M.A., Abdullah S., Hamdan A.R., Kendall G., Qu R. (2007) Solving a Practical Examination Timetabling Problem: A Case Study. In: Gervasi O., Gavrilova M.L. (eds) Computational Science and Its Applications – ICCSA 2007. ICCSA 2007. Lecture Notes in Computer Science, vol 4707. Springer, Berlin, Heidelberg

[2] Abdullah, Salwani. Heuristic approaches for university timetabling problems. Diss. University of Nottingham, 2006.

8 Appendix

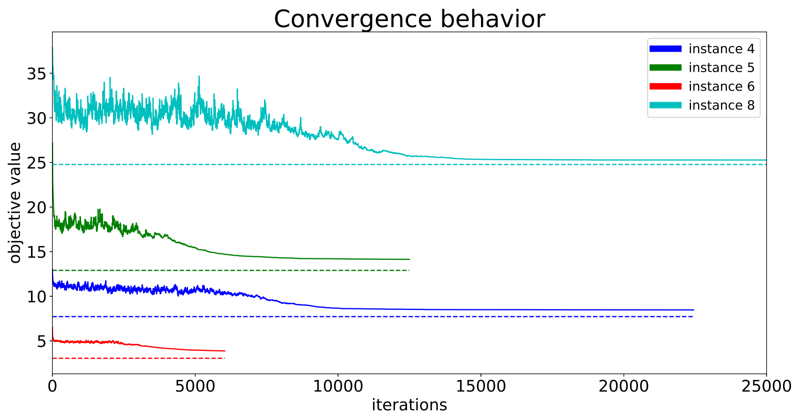


Figure 1: Convergence of the objective value for different instances

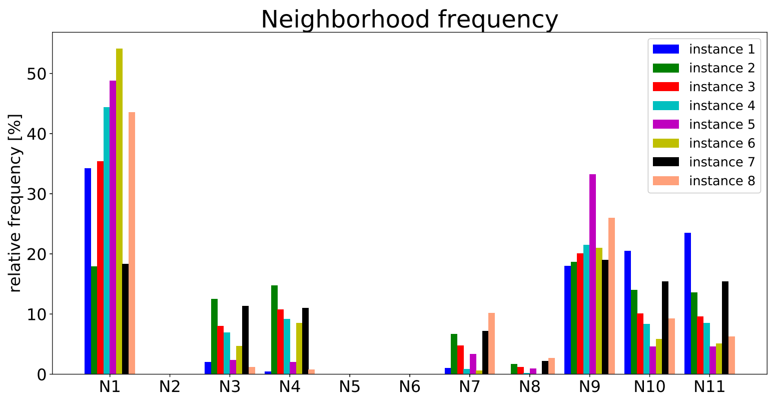


Figure 2: Usage distribution of neighborhood structures

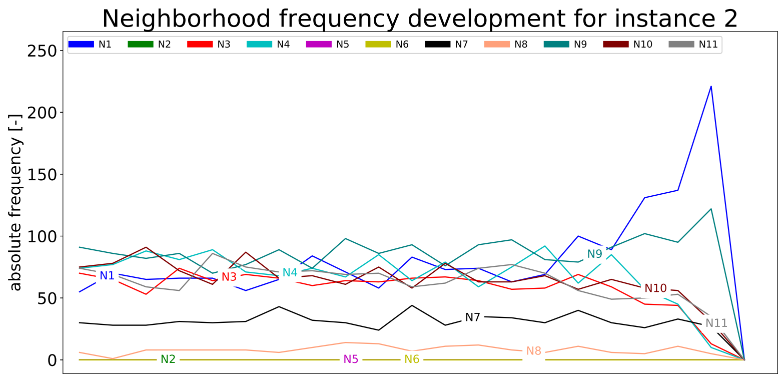
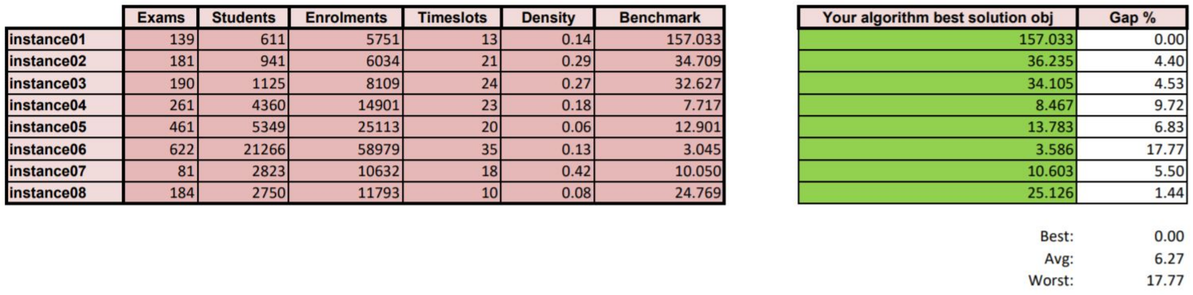


Figure 3: Usage distribution of neighborhood structures over iterations



	Exams	Students	Enrolments	Timeslots	Density	Benchmark
instance01	139	611	5751	13	0.14	157.033
instance02	181	941	6034	21	0.29	34.709
instance03	190	1125	8109	24	0.27	32.627
instance04	261	4360	14901	23	0.18	7.717
instance05	461	5349	25113	20	0.06	12.901
instance06	622	21266	58979	35	0.13	3.045
instance07	81	2823	10632	18	0.42	10.050
instance08	184	2750	11793	10	0.08	24.769

Your algorithm best solution obj	Gap %
157.033	0.00
36.235	4.40
34.105	4.53
8.467	9.72
13.783	6.83
3.586	17.77
10.603	5.50
25.126	1.44

Best:	0.00
Avg:	6.27
Worst:	17.77

Figure 4: Benchmarks