Decision making and optimization - 2019/2020

Group 7:
Calvi Edoardo, Garbarino Matteo, Degola Gabriele,
Atadjanov Olloshukur, Chen Zifan, Thangallapelly Sai Surakshith

Deliverable n.2 - January 2020

1 Introduction

This report addresses a metaheuristic to solve the *Examination Timetabling Problem* (ETP), which involves a series of hard constraints and the minimization of an objective function. At first a Greedy approach is used to generate a set of feasible solutions such that exams with common students are not scheduled in the same time-slot, then a *Genetic Algorithm* (GA) is implemented to reduce the total penalty that must be paid if couples of conflicting exams are scheduled in near time-slots. An exhaustive presentation of the proposed algorithm and of its results follows.

2 Initial Population

One of the first steps of a GA is the generation of a set of initial solutions, called initial population. It is built using a *Greedy* approach, rather than a random generation, as feasibility must be guaranteed. Different approaches have been tested, such as picking exams randomly, or ordering them by the number of conflicts, or by the total number of conflicting students; the best results have been obtained using an hardest-first policy with respect to the number of remaining possible time-slots in which each exam can be placed, based on conflicts.
At the first iteration every exam can be placed in any time-slot, so a randomly chosen exam is placed in a randomly chosen time-slot; then possible time-slots are updated for all the other exams. At every following iteration one of the exams with the lowest number of possible time-slots is chosen, placed in one of those, updating possibilities for the others as described above. If an exam has no remaining options then no feasible solution can be achieved with the current configuration, so the algorithm starts again. The algorithm ends when all exams are placed.
This approach always grants feasibility and provides a solution in a reasonable time. That also guarantees good variability for initial populations.

3 Genetic Algorithm

The previously generated initial population is then improved using a GA, with the aim of reducing its penalty. As we know from the literature[1], the following steps are the computation of each chromosome's fitness, chromosomes recombination and population update, until a fixed number of iterations or a certain amount of time has elapsed.

3.1 Fitness

Given the penalty for each solution, an obvious choice for the fitness would be its inverse as the objective is to minimize it: this yields small changes for fitness values, that can easily go unnoticed in larger populations. A better fitness expression for each chromosome i seems to be $fitness_i = 2^{\frac{referenceValue - penalty_i}{c \times referenceValue}}$, where c is a scaling factor. We tried to use the worst penalty in the population as $referenceValue$ to promote even small improvements. This didn't actually promote good solutions, but rather penalized those outliers that were far worse than the rest. Therefore, we now use the average penalty of the population as $referenceValue.$ This expression brings fast improvements at the expense of the risk to fall in local minima, which is addressed below (see "Escape Local Minima").

3.2 Parents selection

Once the fitness has been computed for all the population individuals (chromosomes), a portion of them is selected for reproduction. The selection is performed in a probabilistic fashion proportionally to the fitness of each chromosome. The probabilistic aspect of the selection allows a wider diversification of the parents and can allow to escape local optima (especially w.r.t. the direct choice of the best individuals).

3.3 Genetic Operators

For the recombination, a modified version of *standard crossover* and three different *mutation* operators are used. Each one affects one or more time-slots, and is chosen in a probabilistic way w.r.t. their contribution to the total penalty. Furthermore, during the test phase each operator provided the best results in a different moment of the execution, so the way they are picked is probabilistic as well and varies over time. In particular, at the beginning of the execution there is a very high probability to apply crossover, which lowers in favor of mutations as time passes.

3.3.1 Crossover

For the crossover a custom method has been implemented, since standard crossover is not suitable for the ETP due to conflicts between exams. Different chromosomes can group exams in different ways, so this is the information of a gene. A random percentage of time-slots (genes) is chosen from the parent chromosomes for the exchange. The method also avoids to place exams in multiple time-slots, and tries to reassign those that have been replaced with the incoming crossing section by adopting the same greedy approach as the initial solution; if any exam has no possible reallocation, its original gene becomes Tabu and needs to be avoided as the crossover restarts. After a certain number of trials, the method fails and simply returns copies of the original parents.

3.3.2 Mutation

Differently from crossover, mutation is applied on a single chromosome and can be useful to escape from local minima since it can innovate stale genes. Three different mutation operators have been developed and each time one of them is applied according to a uniform probability distribution:

- *Exam moving*: it chooses a time-slot, picks one exam among those that could fit in, and moves it. If no movement is possible on that time-slot, the method fails and returns the original chromosome;
- *Time-slots swapping*: two time-slots are chosen randomly and the assigned exams are swapped. This operator always grants feasibility and returns the original chromosome only if the same time-slot is chosen two times;
- *Time-slot destruction*: a time-slot is randomly chosen and all its assigned exams are removed. Then each exam is placed in one of its possible time-slots, favouring the ones

that contribute less to total penalty. Returned solutions are always feasible, as any exam can be at least replaced in its original time-slots.

Each operator provides the best result at a certain execution period, but it varies considerably from one instance to the other.

3.4 Escape local minima

The main defect of the algorithm was that it wasn't strong enough to escape local minima. We devised a method to recognize when the execution was stuck on a given fitness value and not improving for too long. That situation is recognized as a local minima and the population is artificially altered to escape it. More precisely, the best element of the population is extracted and used to generate a new population through a series of mutations, replacing the old one.
The method has shown to be effective, but it is indeed a trade-off between the advantage of escaping a local minima and the acceptance of some worsening solutions. The algorithm often manages to escape the local minima, but it requires a non-negligible amount of time to recover; that's why we also implemented a safety check that forbids the usage of this technique near to the end of the available time.

3.5 Population update

After each recombination the resulting offsprings replace part of the population with an *Elitist* approach preserving the best chromosomes. More specifically we discovered that the best performing configuration was with a small population size (10) because better solutions can impact more on the population, but with an high percentage (90%) of newborn children to compensate. The newly generated individuals will substitute the weakest (lowest fitness) ones of the population keeping the total size constant. The top solution of each iteration is saved locally only if it is strictly better than the previous best one.

4 Conclusion

After several tests, the presented algorithm resulted quite solid. The results are anyway influenced by the algorithm parameters, such as population size and percentage of population updating, but the optimal configuration changes with respect to the instance and it is not possible to find one that fits all.
Moreover, our crossover operator failed to bring relevant improvements, possibly due to the constrained nature of the problem.
The acceptance of worsening solutions in the "escape local minima" technique doesn't negatively affect our result because the writing policy we adopted strictly prevents from saving locally a worse solution w.r.t. the previous, so the only drawback becomes a slower convergence of the algorithm.
We tried to implement a feature consisting in the reintroduction of the solution used to generate the new population, after some iterations. The result was the convergence to same local minima, so the feature has been removed (method "godsMercy()").

References

[1] R. Tadei, F.D. Croce, and A. Grosso. *Fondamenti di ottimizzazione.* Progetto Leonardo, 2005.