A memetic approach for timetabling problem

Andrea Arcidiacono, Simone Santia, Gabriele Cuni, Giulio Alfarano,
Valeria Sorrenti, Marina D'Amato, Michele Apicella

***Abstract*—The Examination Timetabling problem deals with scheduling exams for students while satisfying specified constraints. The problem we approached is a simplification of real instances and it does not consider the number of available rooms or their capacity. Our approach consists of an hybridization of Genetic, Iterated Local Search and recursive algorithms. The algorithm has been tested on a set of real instances and their benchmarks.**

I. INTRODUCTION

The Examination Timetabling Problem aims at assigning each exam to a specific time-slot ensuring some constraints, divided into two types: hard constrains and soft constraints. The hard constraints are the compulsory ones that cannot be violated. In our problem, they guarantee that each exam is scheduled only once during the period and that two conflicting exams – exams that have students in common – are not scheduled in the same time-slot. Soft constraints are not compulsory but they aim at improving the quality of the timetable: in our case, a penalty is applied for each couple of conflicting exams scheduled up to a distance of 5 time-slots.

II. DATA STRUCTURES

The Java multi-thread version of the ETP problem consists of five different classes: `Main`, `Exam`, `Model`, `GeneticAlgorithm`, `IteratedLocalSearch`. The `Model` class is the coordinator between I/O files and the different threads of the algorithm.

Our solution structure is a simple array whose indexes are the exams id's that store the time-slots id's in which the exam is scheduled.

III. ALGORITHMS

A. Genetic Algorithm

Genetic Algorithm is a meta-heuristic that is able to create a whole population of initial solutions instead of a single design point. We implemented our own version of the GA in order to widely explore the solution space and to diversify the population set as much as possible. The population size depends on a parameter *H*, which depends on the difficulty of the specific instance.

$$H = \frac{total\ number\ of\ conflicts}{nExams * nTimeslots}$$

The initial feasible population is generated through the recursive function explained below. A "parent" is chosen randomly among the initial population chromosomes and it generates a "child" with a non-standard random crossover: the cutting sections are chosen randomly and are passed to the Recursive function that, keeping fixed the cutting sections, fills the child with the remaining exams, ensuring the feasibility of the scheduling. We select the new chromosome only if it is better than the worst of the current population.

B. Recoursive Function

Recursive Function follows a semi-deterministic way of scheduling. We have two methods to give a certain order for scheduling: the first sort the exams by the number of conflicts and the second sort the most crowded time-slots in order to reserve the less crowded ones to the most conflictual exams. The recursion tries to fill the solution and, if it doesn't succeed, performs backtracking for a random number of steps until all the exams are scheduled.

C. Iterated Local Search

Iterated Local Search is a heuristic that explores the solution space and widely minimizes the objective function. Given an initial solution, it generates two different neighbourhoods. The first one searches for the best timeslot – in terms of objective function – to switch every single exam to. The latter computes all possible couples of time-slots and then selects the one that minimizes penalty the most. Then, it selects all the exams in the first time-slot that are not in conflict with the exams in the second one. It applies the same procedure to the second time-slot. Finally, it swaps the two exam lists. The algorithm uses a Steepest Descent approach within the first neighbourhood until it reaches a local minimum. We now save this local minimum

and, after swapping the two list of exams with the second neighbourhood, we continue the search using the first neighbourhood.

IV. HYBRIDATION

We will now explain how we performed the hybridation of our algorithms. We fill the initial population of the algorithm with the recursive function until every single chromosome is different from each other. In order to improve diversity of the initial population, we randomly swap the first element of the ordered exams' list with one of the first *k*-elements, where *k* is the number of chromosomes. We remind that this list is ordered by the most conflicting exams, so, because k is really low, we chose this statistic to not disrupt the ratio of the ordered list. Then we start the GA to select the parent and generate the child. Our actual implementation of the GA is a Memetic Algorithm that implements the Iterated Local Search to optimize the child before inserting its local minimum into the population.

V. CONCLUSIONS

In the first chart it's described the evolution of the penalty of the various solutions in time. It can be seen that the objective function has a different trend depending on the time interval in which it is located; the exact time is dependent from the complexity of the instance under consideration. We identified three different interval trends:

- 0 - 60 seconds: the penalty assumes a steep monotonous decreasing behaviour, this is due to the contribution of the Iterated local search, its operation allows the solution to receive a huge drop of penalty generally in the first seconds from the start.

- 60 - 100 seconds: the objective function begins to converge to the local minimum found by the ILS. In this interval the solution found by the aforementioned algorithm is returned to the Genetic Algorithm which performs various crossovers and alterations in order to explore the solution space. Despite this, it cannot find a hugely better minimum or it always returns to the previous minimum which defines the convergence in the graph.

- Over 100 seconds: eventually the trend of the objective function stabilizes to the value of the penalty minimum, so neither the GA or the ILS can improve significantly the function value. It proceeds in this way for a long time; only after about 400 seconds there could be a minimum improvement which is not shown in the chart, but it isn't useful for conclusion purposes.

As it is shown in the second chart, there is still a little gap between our results and the benchmarks in the most difficult instances. With the adoption of other hybridation techniques, our global optimum may improve and reduce the gap.

A. *Charts*