DMO Report:
Exam Scheduling Problem and SA solution

Group 02
Filippo Barba, Adele de Hoffer
Alberto Gennaro, Valeria Turina
Riccardo Vellano, Chiara Vercellino
Matteo Villosio

October 27, 2023

1 Solution Modelling

We have chosen to think at a solution of the problem as a vector of exams, so we have built a class Exam, which is the atom of our model. With this class we were able to store some valuable information, like conflictual exams (i.e those with which there is at least a common student), the weights of the conflicts and his contribution in the global objective function. Thanks to two functions called `read_file_stu` and `read_file_exm` we have created a matrix of conflicts and then we have filled every exam with the information needed. We have then assembled the class Solution, that stores a vector of reference to exams, a vector of times, where there is an indexes correspondence with the previous vector and the objective function value. Every Solution is capable of self verify its feasibility, and to update objective function, time slots of exams and so on.

2 The first problems: Neighborhood Generation and Initial Solution

Talking about initial solution, the best idea we found was to order the exams in terms of connectivity and then try to uniformly distribute exams on the time slots. Here is the pseudo code implementing this idea.

Algorithm 1 Greedy Coloring Methods for Initial Solution

1. Initialization:
   - Initialize **all exams' time-slots** to **-1**
   - **Order exams** by their weight
   - Create vectors that contain a **subset of time-slots** such as {1, 6, 11}, {2, 7, 12} etc called **boxes**
2. Code:

START: All variables to zero and no boxes used.
loop:
while I haven't assigned a feasible time-slot to every exam **do** Take the most important exam and randomly switch some exams already placed
  if It is *possible* to assign to the exam a time slot belonging to the set of available ones in the box **then** do it &
    go to *loop*
  else Reshuffle exams and try to place it again.
  if I couldn't place the exam for a few times **then**

- **if** There are boxes still available add a box (i.e. available time-slots) & **go to** *loop*
- **if** No more boxes available **go to** *START*

We have thought about two ways of exploring the solution space: moving by a not-surely feasible step and then recover the feasibility with easy transformations, or by feasible (and slower) transformation directly. We have chosen the second alternative due to the complexity of obtain a feasible solution from an unfeasible one.

2.1 Directional swapping and Random swapping

The swap idea came from some papers we have read about the Kempe Chain Neighbourhood (KCN) about graph coloring. The idea is simple: if there is a feasible solution with k exams in time slot t_1 and m in time slot t_2, we can change the color (i.e the time slot) without losing the feasibility. We have slightly refined this kind of reasoning.

Algorithm 2 Swapping

1. Select at random two time slots and take all the exams in them
2. for times t_1 and t_2
   - for each exam check if there is at least one conflict with all the others of the other time slot
   - if yes, save it in a vector `toBeSwappedinT[i]`, i = 1,2.
3. for each exam in `toBeSwappedinT[1]`, change their time slot from t_2 to t_1
4. for each exam in `toBeSwappedinT[2]`, change their time slot from t_1 to t_2

2.2 Unscheduling and Rescheduling

The idea is to unschedule a certain number of exams and then trying to casually reschedule them in an available time slot (i.e a time slot where there are no "conflictual" exams). This mutation is not always possible as there is the chance of not being able to reschedule all exams. Despite this, we have noticed that for a small number of exams the probability of failing the rescheduling procedure is very small. The pseudo code is the following:

Algorithm 3 Unscheduling and Rescheduling

1. Unscheduling
   - Pick at random `num_unsched` exams an unschedule them, i.e put a -1 as their time slot
   - Update this information in all exams in conflict with them
2. Rescheduling: for every exam unscheduled unless a certain number of fails occurred:
   - check if it can be rescheduled, i.e if there is an available time slot such that it doesn't host conflictuals
     - if yes, schedule the exam in this time slot
     - else unschedule conflictuals causing problems and augment of 1 the number of fails.
   - Update this information in all exams in conflict with them

2.3 Local Search

We used also this deterministic function to move. The idea is to do improvements based on best time slot for each exam (self optimum vs overall)

Algorithm 4 Local Search

1. Order the exams based on the weight on the overall objective function
2. for each exam check if a move from his actual timeslot can lead to an overall improvement:
   - If yes, make the change
   - otherwise step to the next exam.

3 Simulated Annealing: exponential cooling

In this version we update the temperature of SA in an exponential way ($t = \alpha^k t_0$), we don't use a plateau to keep the temperature constant and when the temperature is too low, we re-heat and explore other regions of the feasible set. The re-heating of temperature is done when $t \leq 0.01 t_0$, and we reset the value of t, that will be again decreased exponentially, to a value of $t_{0_{iter}}$ which corresponds to accept a worsening solution with probability 0.25. We don't reset it to t_0, because we don't want to move too far away from a good region in feasible set.

Algorithm 5 SA algorithm exponential

- Temperature Initialization equal to the previous algorithm

`while` there is still available time

- **Unscheduling**, **Rescheduling** and select the best swap with **Single Swapping**.
- do textbfLocal Search.
- Accept or refuse the new solution according to SA logic
- Update $t = \alpha * t_{0_iter}$
- **if** $t \leq t_0 * 0.01$ **then** $t = t_{0_iter}$

4 Inception Implementation of SA

The idea of this implementation is to have one SA process that evaluates solutions very different and, for every one of them, a littler (in terms of number of changes) SA is finding the "local" best (which is also very quick). The algorithm knows if it is internal or external thanks to a boolean called inception (true if it is internal).

Algorithm 6 Inception

1. Temperature Initialization:
   - Take the Initial Solution from Initialization Step.
   - Create a neighborhood quite different from the first solution and calculate the initial temperature such that the biggest delta of value function calculated in this step lead to a probability of 0.5.
2. `while` there is still time
   - Use actual temperature and initial temperature to calculate number of mutation to do in unscheduling.
   - Do unscheduling, rescheduling
   - If `boolean == false`
     - do the directional swapping
     - call the innested SA and do a Local Search until STOP CRITERION reached
   - If `boolean == true`
     - skip this passage and just do one iteration on Local Search
   - evaluate the new solution against the old one with the SA probability technique, then cool down the temperature linearly.

5 Results and Conclusions

With the combination of the two algorithms we reached quite good results, after lots of run on 3-5 minutes we reached an average gap of 6.30% on the benchmarks. We observed that adding local search and greedy approaches on our algorithm made our algorithm slower in terms of iteration, but all our other methods of neighbourhood generation and exploration didn't give good results in terms of objective function minimization. In fact, before ending to this solution we have tried to generate neighbourhood by mutations, by crossover, by multiple swaps and different combinations of this kind, trying to pilot the solution to a minimum though some parameters, like the number of mutation or the selection of exam to mutate and swaps selecting with higher probability exams that did bring higher cost in objective function. Anyway we realized that we had too much randomness on our algorithm and that we needed something more deterministic, like Local Search and best Swaps policy, to actually reduce objective function. In particular we got

- Instance 1: best result= 157.147 best gap of 0.07%
- Instance 2: best result= 36.822 best gap of 6.09%
- Instance 3: best result= 34.135 best gap of 4.62%
- Instance 4: best result= 8.282 best gap of 7.32%
- Instance 5: best result= 14.262 best gap of 10.54%
- Instance 6: best result= 3.454 best gap of 13.45%
- Instance 7: best result= 10.556 best gap of 5.03%
- Instance 8: best result= 25.582 best gap of 3.28%