POLITECNICO DI TORINO

Optimization Methods and Algorithms

Course Assignment

Examination Timetabling Problem: Algorithm Report

GROUP 5:
Pietro Barbiero 252818
Manuel Scurti 251175
Matteo Senese 249896
Chiara Sopegno 244134
Antonio Tavera 243869

A.A. 2017/2018

Evolutionary Tandem Search for ETP

Pietro Barbiero, Manuel Scurti, Matteo Senese, Chiara Sopegno, Antonio Tavera

I. Data Structures and Efficiency

THE Examination Timetabling Problem (ETP) parameters are organized in three data structures: `EXAM`, `STUDENT` and `GRAPH`. The `EXAM` class has five attributes:

- `ExamID`: the unique exam identifier;
- `EnrolledNum`: the number of students enrolled in the exam;
- `Students`: the list of enrolled students;
- `ConflictingExams`: the list of exams clashing with the current one;
- a list of heuristic attributes used by heuristic methods.

The `STUDENT` class, representing the students involved in the problem, has two attributes:

- `StudentID`: the unique student identifier;
- `Exams`: the list of exams for which the student is enrolled.

The `GRAPH` structure represents conflicts between exams using two attributes:

- `Nodes`: the list of nodes, each of them corresponding to one exam. It has two attributes:
  - the corresponding exam object;
  - `Edges`: the list of edges that represents conflicts with other exams.
- `Edges`: the list of links between nodes, representing conflicts between exams. It has two attributes:
  - two `Node` objects representing the two conflicting exams;
  - the edge `Weight` representing the number of students

The `SOLUTION` class is used for the representation of a problem solution. It has six main attributes:

- `Map<Exam,Integer> mapped_sol`, linking each exam with the corresponding timeslot;
- `Map<Integer,Set<Exams>> complex_sol`, linking each timeslot with the set of exams scheduled for the slot.
- `Rank`: the cost of the solution;
- `Feasible`: the number of conflicts;
- `ExamClashing`: the list of conflicting exams;
- further structures used for solution sorting.

The basic step of the used heuristics consists in moving one exam from one timeslot to another. Since the move may modify the value of the objective function, an efficient update of the solution rank is needed. The solution proposed consists in updating the `Rank` property considering only the weights of the edges adjacent to the moved exam. If the exam e is moved, then the new rank r_{new} is computed using the following equation:

$$r_{new} = r_{old} + \sum_{b \text{ in conflict with } e} w_{eb}(2^{|t_{new}-t_b|} - 2^{|t_{old}-t_b|}) \quad (1)$$

where r_{old} is the previous rank, w_{eb} is the weight of the edge between exam e and exam b (so, the number of students involved in the conflict) and $|t_{new} - t_b|$ is the distance between the new timeslot of e and the timeslot of b.

II. Well-Known Algorithms

In this section three well-known metaheuristic algorithms for optimization are presented. These algorithms have been implemented in order to test their performances on seven ETP istances.

A. Simulated Annealing + GC

The proposed Simulated Annealing (SA, [1] [2]) is a version of the well-known SA which starts from a solution provided by a Graph Coloring (GC) algorithm. The GC sorts the exams in descending order according to the degree of nodes. The degree of a node is the number of edges of the node, which is equivalent to the size its conflict exam set. The GC takes as input a fixed number of colors, corresponding to the number of timeslots. It assigns a color starting from the exam with the highest degree. From that node, it tries to use different colors for its neighbors. It continues coloring until all the nodes are colored. If the number of colors is not enough to cover all the nodes, the remaining ones are assigned to the less conflicting timeslot. Finally, the GC provides its solution to the SA. At each iteration, the SA randomly selects a solution close to the current one using a neighborhood generator. The neighborhood is randomly provided using four different generators:

- The first one randomly selects two timeslots. Then, it selects at most n-exams from the first timeslot that are not in conflict with the exams in the second one. It adopts the same procedure starting from the second timeslot. Finally, it swaps the two exam lists.
- The second one randomly selects two timeslots. Then, it selects from the first timeslot the exam with the highest cost that will not be in conflict in the second slot. It adopts the same procedure starting from the second timeslot. Finally, it swaps the two exams.
- The third one selects the timeslot that produces the highest cost. From that timeslot, it selects the exam that generates the highest cost. Finally, it moves the selected exam in the first not-conflicting timeslot.
- The last one randomly selects two timeslots. Then, it selects the exam from the first timeslot that is conflict free in the second timeslot. Finally, it moves the exam in the second slot.

If the new solution is better than the current one it moves to the new solution. Otherwise, it draws a random number ρ in $[0, 1]$ and it accepts the move to the new worse solution if ρ is less than $e^{-\delta/T}$. During the search, the temperature T is

progressively decreased from an initial positive value to zero, affecting the probability of moving to a worse new solution, which is progressively changed towards zero. The temperature update guarantees the convergence of the algorithm to a good solution.

B. Tabu Search

The Tabu Search (TS, [3] [4]) algorithm enhances the performance of local search relaxing two of its rules. First, at each step worsening moves can be accepted if no improving move is available. This relaxation is useful when the algorithm is stuck in a local minimum. In addition, if a potential solution has been previously visited within a certain short-term period, it is marked as *tabu* (forbidden) so that the algorithm does not consider that possibility repeatedly. For the ETP, the TS is based on three criteria:

- A special rank r_s, designed in order to compare feasible and unfeasible solution:

$$r_s = r + \eta C \tag{2}$$

  where r is the feasible solution rank, η is a penalty coefficient and C is the number of conflicts contained in the solution.
- The Tabu List, containing an `HashSet` with the most recent visited solutions (instead of moves). The use of tabu-solutions instead of tabu-moves is justified by experimental evidences.
- The Aspiration Criterion, which allows the move to a tabu-solution $s(i)$ if it is sufficiently close to the best solution:

$$|r_{\text{best}} - r_{\text{i}}| \leq t_{AC} \Rightarrow \text{move to } s(i) \tag{3}$$

  where t_{AC} is an heuristic threshold.

In some instances, when the TS approaches a local minimum, it tends to go outside the feasible search space. In this case, the number of conflicts can increase without control. Therefore, a fixing method has been implemented. It creates a list of pairs of exams in conflict. For each pair, it tries to move one exam at a time. After all the attempts, it returns the fixed (or partially fixed) solution. However, given the computational burden of the procedure, it runs only when the number of conflicts is limited (e.g. 5).

C. Genetic Algorithm

In a Genetic Algorithm (GA, [5] [6] [7]), a population of candidate solutions is evolved toward better solutions. The evolution starts from a population of randomly generated individuals. Some individuals are stochastically selected from the current population and assigned to a list called `Parents`. After that, the GA alters the properties of the `Parents` solutions drawing two random numbers $p_1, p_2 \in [0, 1]$:

- If p_1 is greater than a fixed quantity $p_C \in [0, 1]$, then the elements in the `Parents` list exchange each other's properties (exam-timeslot assignments) in a crossover process.
- If p_2 is greater than a fixed quantity $p_M \in [0, 1]$, then the elements in the `Parents` list mutate one of their properties.

The rank is evaluated for each element in the `Parents` list. Finally, a subset of the current population is selected to form the new generation of candidate solutions in the next iteration of the algorithm.

III. Evolutionary Tandem Approach

An hybrid approach is an ensemble of multiple metaheuristics. The idea is to use different strategies that helps each others during the search. Concerning the algorithms presented in Sec. II, both the SA and the TS work well if they start from a solution which is near to a good solution, otherwise their local search is useless. The developed Evolutionary Tandem Approach (ETA) is an hybrid approach is designed in order to take into account this objective. It is composed of a GA that provides a huge amount of different solutions both to a SA and a TS algorithm.

A. Fixing Phase

Making a feasible solution from an unfeasible one is a crucial step in order to provide good feasible solutions to local search algorithms. The Fixing Phase (FP) of the ETA is designed to achieve this objective [8]. During the FP, the algorithm sorts exams according to three parameters:

- The Conflict Grade (CG): the number of conflicting exams with the one under analysis.
- The Highest Cost (HC): the cost of scheduling the exam given the actual uncompleted timetable.
- The Saturation Degree (SD): the number of remaining slots in which the exam can be scheduled without creating a new clash.

After the sorting phase, the algorithm starts to fill up the timetable with the remaining exams following the sort order, proceeding until the list is empty. The objective is to schedule the most difficult exam to the minimum penalty period at each iteration.

B. Adaptive-Supervisor and Self-Adaptation

Since in high dimensional spaces the instances can have very different shapes and complications, approaches with fixed parameters risk to work well only for a subset of instances, lacking of flexibility. An adaptive algorithm tackles this problem modifying its parameters according to the characteristics of the provided instance [9]. The ETA has two adaptive methods:

- The Adaptive Supervisor (AS) is a high-level adaptive method. It has two objectives:
  - It tries to *sniff* the complexity of the current instance using a simple heuristic:

$$h = \frac{\#\text{exams}}{\#\text{timeslots}} \tag{4}$$

    When h is high, the AS *sniffs* that the current instance could be complex. Indeed there are few

timeslots with respect to exams, leading to the possibility of having a huge number of conflicts. This means that it could be hard and slow even finding a feasible solution. On the other hand, if h is low, there are more available timeslots and finding a feasible solution should be easier. For hard instances (high h), the AS reduces the population size of the GA, because the generation of each feasible individual can be very expensive. Instead, for easier instances (low h), the AS increases the population size, guaranteeing a greater biodiversity among each generation.

- When it senses that the algorithm is stuck (the best solution does not change for some iterations), it perturbs all the GA, SA and TS parameters.

- The Self-Adaptive Mutation (SAM) is a method internally used by the population individuals. When a solution is selected for the mutating process, the amount of mutating elements is computed as:

$$m = e^{-\lambda} \tag{5}$$

where λ is the self-adaptation learning rate and m is the fraction of properties of the solution that mutates. The learning rate decreases generation by generation, leading to a decrease of the mutation rate. However if a solution does not improve, the probability of being selected for the following generations is decreased by the GA. Therefore, the SAM *shakes* the learning rate λ, hoping that an higher mutation rate will increase the solution rank.

IV. Results

The algorithms presented in Sec. II and the implemented ETA have been tested using seven instances. The ETA outperforms the GA, the SA and the TS. Concerning the adaptation, the ETA has been compared with its adaptive counterpart. Fig. 1 shows a comparison between the two versions. The x-axis represents the time in minutes since the algorithm start. The y-axis represents the mean distance to the best solution in literature over the seven instances on multiple attempts. The adaptive ETA is constantly better (on average) than the static version. Fig. 2 shows the first derivative of the mean distance with respect to time. From these figures, it can be observed that:

- On average, the ETA immediately finds a slightly better set-up, which leads to better solutions. Additionally, the ETA keeps this advantage in the following iterations. This observation may confirm that the Adaptive Supervisor *sniffing* increases the performance.
- The fact that the two derivatives are approximately identical may suggest that the adaptation does not affect the improvement rate during the search. However, the effort of improving from a better solution has not being investigated.

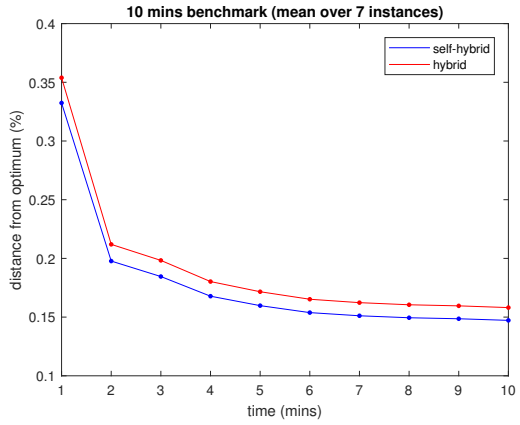

Fig. 1. Comparison between the Evolutionary Tandem Approach and its adaptive version. Mean distance to the best solution in literature.

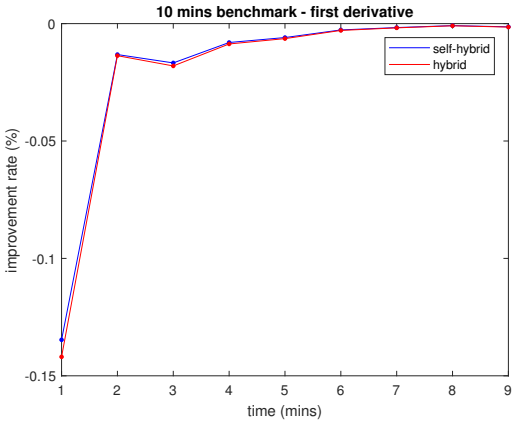

Fig. 2. Comparison between the the Evolutionary Tandem Approach and its adaptive version. First derivative of the mean distance.

References

[1] P.J. van Laarhoven, E.H. Aarts *Simulated Annealing: Theory and Applications*, Springer.

[2] Aycan, Ayav *Solving the Course Scheduling Problem Using Simulated Annealing*, Advance Computing Conference, 2009. IACC 2009. IEEE International.

[3] Glover, Fred, Laguna, Manuel *Tabu Search*, Springer.

[4] Di Gaspero, Schaerf *Tabu Search Techniques for Examination Timetabling*, Lecture Notes in Computer Science.

[5] Eiben, Agoston, Smith, James *Introduction to Evolutionary Computing*, Springer.

[6] Zhang et al. *A Genetic Algorithm Using Infeasible Solutions for Constrained Optimization Problems*, The Open Cybernetics & Systemics Journal 8(1):904-912.

[7] https://www.tutorialspoint.com/genetic_algorithms/.

[8] Pillay, Banzhaf *A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem*, European Journal of Operational Research.

[9] Kramer *Self-Adaptive Heuristics for Evolutionary Computation*, Springer.