Solving the Curriculum-Based Course Timetabling problem using Iterated Local Search algorithm

Gresa Neziri, Artinë Rexhepi
January 2019

1. Problem description

This problem is about Course timetabling (CTT) that consists of the weekly scheduling of the lectures of a set of university courses within a given number of rooms and time periods, satisfying various constraints due to conflicts and other features [1].

1.1. Basic entities

The problem consists of the following basic entities:

Courses and Teachers. Each course has a unique id and consists of a fixed number of lectures to be scheduled in distinct periods, it is attended by a given number of students, and is taught by a teacher.

Days, Timeslots, and Periods. There is a number of teaching days in the week that are defined in input file. Each day is split into a fixed number of periods, which is equal for all days. A timeslot is a pair composed of a day and a period. The total number of scheduling timeslots is the product of the days times the day periods.

Curricula. A curriculum has a unique id and consists of a group of courses such that any pair of courses in the group have students in common. Based on curricula, we have the conflicts between courses and other soft constraints.

Rooms. Each room has a unique id, a capacity that is the number of available seats, and a location expressed as an integer value representing a separate building (in our case the building is not used). Some rooms may not be suitable for some courses (because they miss some equipment).

1.2. Constraints

Lectures: All lectures of a course must be scheduled. A violation occurs if two lectures are in the same timeslot or a lecture is not scheduled. We need to apply a break period if the lecture's length is longer than 2. Number of breaks that should be added are defined in cost components.

Room Occupancy: Two lectures cannot be in the same room in the same timeslot.

Conflicts: Lectures of courses taught by the same teacher or in the same curriculum must be all scheduled in different timeslots.

Availability: If the teacher of the course is not available to teach that course at a given timeslot, then no lecture of the course can be scheduled at that timeslot. Given timeslots that a teacher is not available are constraint of type period.

Room Capacity: For each lecture, the number of students that attend the course should be less than or equal the number of seats of all the rooms that host its lectures.

Room Suitability: Some rooms may be not suitable for a given course because of the absence of necessary equipment. These rooms are given in constraints of type room and that course can not take place on them.

Windows (Compactness): Lectures belonging to a curriculum should not have time windows (periods without teaching) between them. For a given curriculum we account for a violation every time there is one windows between two lectures within the same day.

All the above cost components are hard constraint, except for Room Capacity and Windows which are soft.

1.3. Fitness function using cost components

In the table below is shown our problem formulation including all hard constraints and cost components that are applicable in our solution..

Cost Component	*PR2*
Lectures	H
Conflicts	H
RoomOccupancy	H
Availability	H
RoomCapacity	1
Windows	1
RoomSuitability	H
All lectures of individual courses should be taught in block	H
Number of periods per lecture	3
Number of breaks per block of lecture	$f_{PR2}(x)$

Table 1 - Fitness function cost components

$x - Number\ of\ lectures$

$f(x) - Number\ of\ breaks\ per\ block\ of\ lecture$

$$f_{PR2}(x) = \begin{cases} \frac{x-1}{2} & for\ odd\ x \\ \frac{x-2}{2} & for\ even\ x \end{cases}$$

When all the hard constraints are met, we achieve a feasible solution. The score is equal to the sum of all the costs of components that are soft constraints. The ideal score to try to achieve is 0.

1.4. Solution of the problem

The solution of the problem is an assignment of a timeslot (day and period) and a room to all lectures of each course.

2. ILS (Iterated Local Search) algorithm

ILS is a metaheuristic which is a clever version of Hill-Climbing with Random Restarts. The algorithm works in such a way that it tries to hill-climb in the space of local optima. The algorithm finds a local optimum and then looks for another local optimum that is nearby, if it is possible adopts the last one. Then it finds a new nearby local optimum and so on [2].
ILS maintains a "home base" local optimum of sorts and selects new restart locations that are somewhere in the region of the "home base" local optimum. When it discovers a new local optimum, it decides whether to keep the current "home base" local optimum, or to adopt the new local optimum as the "home base".

Below is the pseudo code of the Iterated Local Search:

Algorithm Iterated Local Search (ILS) with Random Restarts ()
total time ← total running time of the algorithm
T← distribution of possible time intervals in near future
S← some initial random candidate solution
H← *S*
Best← *S*
repeat
    time← random time in the near future, chosen from *T*
    repeat
        R← *Tweak(Copy(S))*
        if Quality(R) > Quality(*S*) **then**
            S ← *R*
    until S is the ideal solution, or *time* is up, or we have run out of *total time*
    if Quality(*S*) > Quality(*Best*) **then**
        Best ← *S*
    H ← NewHomeBase(*H, S*)
    S← *Perturb(H)*
until Best is the ideal solution, or *time* is up, or we have run out of *total time*
return Best

The algorithm does hill-climbing until time is up and then decides on whether to get the new local optimum solution or to go back to the current "home base". Then from the new "home base" makes a perturbation to jump to a new hill.
Perturbation is meant to make an escape from the current local optimum, but not as big as to be completely randomized. While the goal of the NewHomeBase is to pick new starting locations. The algorithm adapts to the new local optimum [2].

$$NewHomeBase(H,S) = S$$

Or could only use the new local optimum if it's of equal or higher quality than the old one.

$$NewHomeBase(H,S) = \begin{cases} S & if\ Quality(S) \geq Quality(H) \\ H & otherwise \end{cases}$$

3. Problem Solution

We solved the described problem using ILS (Iterated Local Search) algorithm into three phases of implementation:

- Initial Solution
- Operators and Fitness Function
- Perturbation and New Homebase.

Code implementation is done using C# programming language in Visual Studio 2017.

3.1. Initial Solution

The initial solution is randomly generated taking care every hard constraint to not be violated. For each course sorted by curriculum that belongs to is chosen a random valid room with good capacity and a random day. After that, for each period on that day is checked if is a valid one. When a valid period is chosen, then an assignment is created and is added on the list of assignments. The pseudocode is given below.

Pseudo-Code – The method GenerateSolution() which returns List of Assignments

```
For each course in sorted list
      Choose random day
     Do
             Choose room with good capacity
      While valid room is chosen
      For each period
             If hardconstraint are not met: break
      End
      Add assignment into list (course,room,timeslot)
End
return List of assignments
```

3.2. Operators and Fitness Function

We have implemented two operators:

a) SwapRoomMutation

The aim of this operator is to improve the score by reducing room capacity as soft constraint. The pseudocode for this operator is as below:

Pseudo-Code – The SwapRoomMutation() which mutate initial solution

```
Choose random course from assignment list where capacity of room is lower than size
of students
Choose another random course with the same criteria
If hard constraints are not met and the room of each course is capable
      Swap rooms
End If
```

Example of this mutation is shown below:

	CourseX	*CourseY*
Room R	rX	rY
R capacity	120	160
Students	150	110

After Mutation ⇨

	CourseX	*CourseY*
Room R	rY	rX
R capacity	160	120
Students	150	110

If this mutation is applied, it will always improve the total score.

b) ChangeTimeslotMutation

The aim of this operator is to improve score by trying to eliminate windows as soft constraints. The pseudocode for this operator is as below:

Pseudo-Code – The ChangeTimeslotMutation() which mutate initial solution

```
1: Choose random course from assignment list
2: Choose random timeslot
3: If hard constraints are not met
4:      Change timeslot
5: End If
```

Example of this mutation is shown below:

	CourseX
Day	0
Period	15

After Mutation ⇨

	CourseX
Day	4
Period	10

Applying this mutation does not promise always better results than initial solution.

Fitness function is implemented by summing up the costs as shown in table1.

3.3. Perturbation and New Homebase

Perturbation is meant to make an escape from the current local optimum, but not as big as to be completely randomized. Implementation of this function is made by applying the operator ChangeTimeslotMutation 10 times in the candidate solution. A simple pseudocode is given as below:

Pseudo-Code – The Perturb() which change candidate solution

```
1:  i=0
2: while i<10
3:          ChangeTimeslotMutation(), increment i
```

New homebase function is implemented to work exactly as described in section 2.

4. Experiments and results

Experiments are done using two input instances *UP-FME-AS2017* and *UP-FME-SS2018*. The main characteristics of the instances are shown in the table below:

	UP-FME-AS2017	***UP-FME-SS2018***
Courses	310	305
Rooms	27	27
Curriculums	101	65
Days	5	5
Periods per day	48	48
Period constraints	69	305
Room constraints	310	305

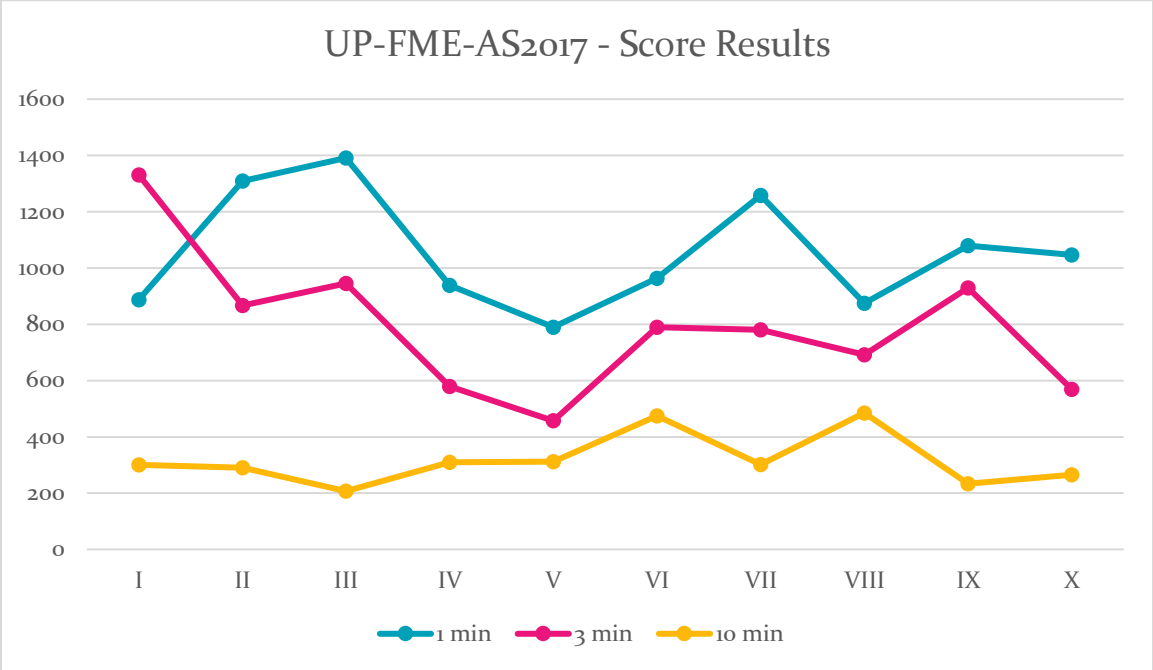
Table 2 - Characteristics of the instances

For each instance there are chosen three time intervals for experiments: 1, 3 and 10 minutes. How this has affected total score is shown in tables below that contains 10 executions for every time interval that means in total 30 different executions for each instance.

UP-FME-AS2017										
	I	II	III	IV	V	VI	VII	VII	IX	X
	1 minute									
Initial	1080	1875	1888	1650	1157	1526	1772	1409	1894	1779
After ILS	**887**	**1309**	**1391**	**938**	**790**	**964**	**1258**	**875**	**1080**	**1047**
	3 minutes									
Initial	2096	1599	1548	1351	1224	1373	1338	1385	1541	1457
After ILS	**1331**	**867**	**945**	**576**	**457**	**789**	**780**	**492**	**929**	**569**
	10 minutes									
Initial	1502	1421	1201	1747	1632	1332	1425	1654	1203	1102
After ILS	**300**	**290**	**207**	**310**	**312**	**475**	**302**	**485**	**234**	**265**

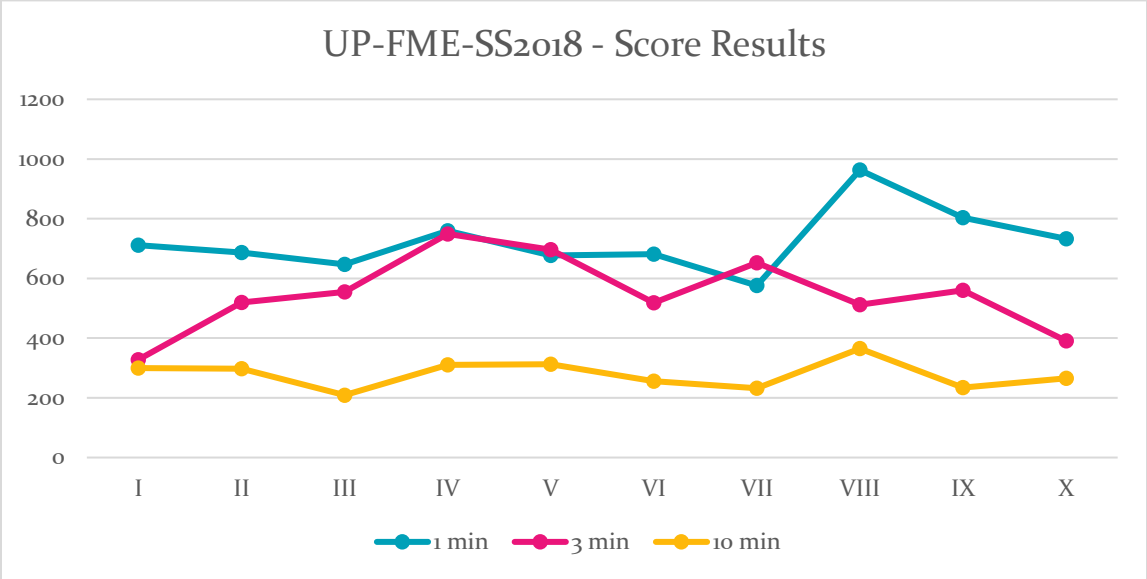
Table 3 - Results of score for the first instance

In the chart below is shown how the time depends on overall score of solution for UP-FME-AS2017 instance.



UP-FME-SS2018										
	I	II	III	IV	V	VI	VII	VII	IX	X
	1 minute									
Initial	1174	931	993	1173	1081	996	918	1383	1235	1402
After ILS	**711**	**687**	**647**	**760**	**677**	**681**	**576**	**963**	**804**	**733**
	3 minutes									
Initial	1212	1203	1005	1235	1272	1065	1098	977	1297	881
After ILS	**328**	**519**	**555**	**749**	**696**	**518**	**780**	**512**	**560**	**391**
	10 minutes									
Initial	1022	1102	985	996	800	1065	1201	1304	1074	986
After ILS	**300**	**298**	**209**	**310**	**312**	**256**	**232**	**365**	**234**	**265**

Table 4 - Results of score for the second instance



After applying ILS algorithm, better results are achieved for both instances. The time interval of execution of algorithm is very influential in total score. More time, better score and this is shown in the charts above for both instances.

Best result achieved for *UP-FME-AS2017 instance* is ***207*** whereas for *UP-FME-SS2018* is ***209.*** Taking in consideration these results, the algorithm has managed to show very good results

5. References

[1] Bonutti, A., De Cesco, F., Di Gaspero, L., Schaerf, A.: Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Ann. Oper. Res. 194(1), 59–70 (2012)

[2] Sean Luke, 2013, Essentials of Metaheuristics, Lulu, second edition, available for free at http://cs.gmu.edu/~sean/book/metaheuristics/