MISTA 2015

Thesis Defense Timetabling

Michele Battistutta · Sara Ceschia · Fabio De Cesco · Andrea Schaerf

Abstract The *thesis defense timetabling* problem consists in composing the suitable committee for a set of graduation sessions and assigning each candidate to one of the sessions.

In this work, we define the problem formulation that applies to some Italian universities, and we provide two solution methods based on local search and constraint satisfaction, respectively. In addition, we perform an experimental analysis and comparison on both real-world and artificial instances.

1 Introduction

The thesis defense and the graduation ceremony are ineludible activities of the management of a university. Large departments may have many students graduating at the same time, and consequently need to split the graduation procedure into several sessions, with separate committees and possibly running in different days.

The corresponding timetabling problem consists in both assigning each candidate to one session and composing the suitable committees, satisfying various constraints and objectives.

This is an interesting NP-hard problem that, up to our knowledge, has been little studied in the relatively-large literature on educational timetabling problems (see [6, 8, 11] for surveys).

In this work, we propose a problem formulation obtained by modeling the real-world problem of an Italian university (Department of Psychology of the University of Milano-Bicocca). We develop both a MiniZinc [9] model and a local search technique to solve the problem. We also develop an instance generator that

Michele Battistutta, Sara Ceschia, and Andrea Schaerf
DIEGM, University of Udine, Via delle Scienze 206, 33100 Udine, Italy
E-mail: {michele.battistutta,sara.ceschia,schaerf}@uniud.it

Fabio De Cesco
EasyStaff s.r.l., Via Adriatica, 278 - 33030 Campoformido (UD), Italy.
E-mail: fabio@easystaff.it

produces realistic cases, in order to have enough instances available to test our solvers. We compare the results of some available CP-based MiniZinc engines and our local search solver on a set of instances produced by our generator and a few real-world ones, on a fixed timeout (5 minutes).

The outcome has been that, on the given dataset and timeout, the local search solver clearly outperformed the MiniZinc engines working on the proposed model.

All instances and results are available on the website `https://bitbucket.org/satt/tdtt-instances`.

2 Problem formulation

The thesis defense timetabling (TDTT) problem consist on the assignment of graduation candidates (students) to different sessions. In addition, for every session the committee must be composed by selecting appropriate faculty members. Usually there are two sessions per day (morning and afternoon) and the duration depends on the number of students assigned. If these are not enough to schedule all students, the university adds new sessions, typically in parallel to the ones already scheduled.

Students must be assigned exactly to one session, whereas faculty members can participate to more than one committee (or none). The number of members of the committee is not fixed, but it is bound by limits prescribed by university rules and traditions. In addition, there are limits on the composition of the committee in terms of qualified members, such as full and associate professors.

The presence of students and faculty members is interconnected because each student has a supervisor, that must be in the committee examining the student. In addition, the committee should include also an *opponent* (or *challenger*) that is a faculty member with expertise in the area of the thesis.

Summarizing, the main entities of the problem are:

Faculty members: The list of university staff that can be assigned as committee members. They are characterized by their role (e.g., full professor) and a list of sessions which they are unavailable to attend.

Students: For each student, it is given his/her supervisor and a list of the potential opponents. Supervisors and opponents are necessarily faculty members.

As customary, constraints are divided into hard and soft ones. The former must be always satisfied, the latter compose the objective function.

The hard constraints are:

Supervision: For each student, the supervisor must be present at the session assigned to the student.

Students per session: The number of students assigned to a session must be less than or equal to the given maximum.

Overlapping sessions: In case that two sessions overlap in time, no member can be assigned to the committee of both sessions.

Committee composition: Each committee must be formed respecting the minimum and maximum number of professor for each *academic level* (or role). In detail, there are four levels, which are (starting from the highest): full professor, associate professor, assistant professor, and external teacher.

Minimum and maximum values for *Committee composition* must be interpreted as minimum and maximum number of members with the given level or a higher one. For example, if we require a minimum of four associate professors, this can be also fulfilled with two associate professors and two full professors. Consequently, the limits for the lowest level represent the limits for the total number of members.

Notice that it is not explicitly defined a minimum number of students per session because the number of sessions is already regulated to fit the given number of students.

The soft constraints are:

Multiple duties: Each time a faculty member has to be in more than one committee, a penalty is assigned. In order to avoid high loads, the penalty of the violations is quadratic. That is, if p is the number of presences of a faculty member, the associated penalty is $(p-1)^2$.

Opponent's presence: For every student, there must be at least one of the suggested opponents in the corresponding session. Each student without opponent counts as one violation of this constraint.

Regarding the size of real cases, they can get to 20 sessions, and 150 candidates and 150 faculty members. The typical size of a committee is between 7 and 10 members. The number of possible opponents for each student normally ranges between 1 and 3. However, in some rare cases, it is possible that no opponent is suggested for a student. In this situation, all faculty members are considered suitable opponents for the student.

3 Computational complexity

We now prove that the decision problem underlying TDTT is NP-complete. The following proof considers only one single session, in which all students are assigned to it. This means that the subproblem of selecting the appropriate faculty members for a single session is already NP-complete.

Consider a *set covering* problem with universe U, with $|U| = m$, the sets $S_1, \ldots, S_n \subseteq U$, and an integer c, with $c < n$. It is well known (see [4], problem SP4) that checking if there is a set of c sets that covers the whole U is an NP-complete problem.

We now show that we can build an instance of TDTT such that it has a 0 cost solution if and only if there is a covering of U composed of c sets.

We create a TDTT instance with m students and $n+1$ faculty members, such that all students have as supervisor the faculty member number $n+1$. For each faculty member i from 1 to n, we assign her/him as opponent of all students in S_i. All faculty members are available for the session. Finally we set a maximum number of members equal to $c+1$ (one spot is for the supervisor $n+1$). It is easy to see that if there is a committee in which all students have an opponent, this corresponds to a selection of the sets that covers all elements in U.

4 Solution techniques

4.1 Local search

We developed a solver based on local search that works on a reduced search space: only students (and consequently supervisors) and opponents are assigned to sessions. Then the procedure has a post-processing step that uses a greedy technique to complete the committee with available members, satisfying committee composition constraints.

The main features of our local search algorithm are:

Search space: The search space consists in the assignment of all students to any session, and the selection of one opponent to each of them. The supervisor and the selected opponent are inserted in the committee of the candidate. Infeasible solutions are part of the search space and the violation of hard constraints are penalized in the cost function with a high weight.

Initial solution: The algorithm start from an initial solution in which students are assigned to a random session and the opponent is chosen randomly from the list of potential ones.

Neighborhood relation: The neighborhood relation is composed by the union of four different moves:

1. Assign the student to a different session and/or change the assigned opponent.
2. Swap the sessions of two students.
3. Assign to a different session a group of student that are assigned to the same session and have the same supervisor.
4. Swap the sessions for two groups of students: each group is formed by students in the same session and with the same supervisor.

In neighborhoods 2-4 the selected opponent remains unchanged, and she/he moves to the new session along with the student.

Neighborhoods 3 and 4 were added for a better space exploration since the solver, affected by the constraint of the maximum number of members for committee and by the objective of limit the presence of each member, tends to reach solutions where students with the same supervisor are assigned to the same session. All four types of move have the same probability of being chosen when the algorithm generates a random move.

Since in our search space, only the supervisor and the opponent are assigned to the committee, it is normally necessary to add other members to complete the committee reaching the minimum number. In this case, the solution is processed with a greedy algorithm that assigns to the committee the missing members minimizing multiple duties.

As metaheuristic technique, we use Simulated Annealing, in its “standard” version as proposed in [1].

4.2 Constraint programming

We developed a constraint model in MiniZinc that uses as decision variables:

– an array of integers that assigns to each student the selected session,

– an array of sets that assigns to each session the set of faculty members that compose the committee.

```
array [1..Candidates] of var 1..Sessions: StudentSession;
array [1..Sessions] of var set of 1..FacultyMembers: SessionMembers;
```

Using set variables for the committee composition makes automatically satisfied the constraint that all members must be distinct. In addition, the constraints regarding simultaneous sessions can be easily expressed using the built-in `disjoint` operator between sets.

The constraints about the composition of the committees based on the level of the participants are expressed as follows, where the component `1` of the elements of the array `LimitMembersForLevel` is the minimum and the component `2` is the maximum.

```
constraint forall (s in 1..Sessions, lv in 1..Levels)
    (((sum (p in 1..FacultyMembers)(AcademicLevel[p] <= l
     /\ p in SessionMembers[s]))
        >= LimitMembersForLevel[lv,1]) /\
     ((sum (p in 1..FacultyMembers)(AcademicLevel[p] <= l
     /\ p in SessionMembers[s]))
        <= LimitMembersForLevel[lv,2]));
```

In order to define the objective function to be minimized, we introduce new decision variables that are functionally related to the main ones. We show here the ones defined for the objective component *Opponent's presence* (corresponding ones are used for *Multiple duties*).

In detail, we introduce the following variables, where `missing_opponent` is the variable that is included in the objective function with the given weight.

```
array [1..Candidates] of var 0..FacultyMembers: NumOpponents;
var 0..Candidates: missing_opponent;
```

These new (redundant) variables are linked to the main ones by the following constraints

```
constraint forall (s in 1..Candidates)
     (NumOpponents[s] = sum (op in Opponents[s])
       (op in SessionMembers[StudentSession[s]]));

constraint
   missing_opponent = sum (s in 1..Candidates)
        (NumOpponents[s] == 0 /\ card(Opponents[s]) > 0);
```

The full model is available along with the instances and the results on the website.

5 Experimental analysis

The local search code is written in C++, using the framework EasyLocal++ [3], compiled using `gcc` v. 4.9.1, and parameter tuning has been performed using JSON2RUN [13].

All experiments ran on an Ubuntu Linux 13.04 machine with 16 Intel® Xeon® CPU E5-2660 (2.20 GHz) physical cores, hyper-threaded to 32 virtual cores. A single virtual core has been dedicated to each experiment.

5.1 Instances

We used three real-world instances, coming from the Department of Psychology of the University of Milano-Bicocca. In addition, we developed an instance generator parametrized by the number of sessions.

The generator randomly selects the number of students and faculty members based on the limits fixed for the composition of sessions. The maximum number of students for session is a value between 8 and 10 and is used along with the number of sessions to define the total number of students, so that each session has at least 6 students. The generator also pairs a random number of session for the simultaneous constraint, the number of pair simultaneous sessions in a generated instance ranges from 0 to one fourth of the number of sessions.

The committee size and composition are fixed: the committee goes from 7 to 10 members with at least 1 faculty member of rank 1 and three of rank lower or equal to 2. We choose to use these fixed values since they were common for all the real cases that we analyzed. A student normally has up to 3 opponents.

In order to create realistic data for the opponents, all faculty members are assigned to an area. Once the supervisor is randomly selected, the potential opponents are selected among the faculty member of the area of the supervisor and the two adjacent ones. If no member, besides the supervisor, exists in these three areas, the student is left without suggested opponents, so that the opponent can be anyone in the committee.

We have created 20 instances with a number of sessions ranging from 5 to 15, and experiments are performed on these instances. Instances are written in MiniZinc data format and they are available on the website (`https://bitbucket.org/satt/tdtt-instances`), along with the MiniZinc model.

5.2 Tuning

Simulated Annealing has several control parameters: the cooling rate (α), the number of neighbors sampled at each temperature (N), and the starting and final temperatures (T_0 and T_{min}). In order to find the best configuration of these parameters using a statistically-principled approach, we resort to the F-Race procedure [2] with a 95% confidence.

With the aim of equalizing approximately the running times for all different configurations, we fix the total number of iterations $I = 10^6$ and compute the parameter N from the others so that the total I is always the same. This results in an average running time of 5 minutes on our machine.

Preliminary results demonstrate that our solution method is not sensitive to small variations of the cooling rate, thus we decided to set $\alpha = 0.99$ and focus on the other control parameters T_0 and T_{min}.

We test 30 different configurations generated according to the *Hammersley point set* [5] for the ranges whose bounds are $T_0 = [10^0, 10^2]$ and $\rho = [10^1, 10^3]$, with $\rho = T_0/T_{min}$. The winning configuration turned out to be $T_0 = 1.18$ and $T_{min} = 0.104$. All the following experiments have been performed using these values for the parameters.

Instance	MiniZinc *fd*	Gecode	Opturion CPX	Local search avg	Local search best
gtt-art01	–	–	210	3.7	1
gtt-art02	346	386	293	1.2	0
gtt-art03	169	204	167	0.3	0
gtt-art04	347	–	258	6.2	2
gtt-art05	–	–	268	0.2	0
gtt-art06	–	–	265	0.5	0
gtt-art07	–	–	333	1.7	0
gtt-art08	–	–	296	1.1	0
gtt-art09	74	–	67	4.1	2
gtt-art10	–	–	310	1.5	0
gtt-art11	394	450	277	2.0	0
gtt-art12	88	–	118	1.0	1
gtt-art13	227	–	219	8.2	5
gtt-art14	–	–	342	3.3	2
gtt-art15	–	–	224	11.0	7
gtt-art16	461	447	417	7.3	4
gtt-art17	–	–	288	1.7	0
gtt-art18	78	123	111	7.0	6
gtt-art19	–	314	302	1.6	0
gtt-art20	402	–	326	6.4	4
gtt-real01	–	567	289	61.1	59
gtt-real02	68	–	80	0.0	0
gtt-real03	–	904	440	54.1	52

Table 1 Comparison between MiniZinc engines and local search on the testbed.

5.3 Comparison of results

We experimented several engines for MiniZinc. Among those available in the MiniZinc distribution v.2.0 only *fd* and *fdmip* support the *set* construct which is used by our model. However, in Table 1 we decided to show only the results obtained by *fd*, which is the default evaluation algorithm for MiniZinc, given that they have the same performances. In addition, we compare our results with Gecode v.4.3.3 [12], which is a pure constraint programming (CP) solver, and Opturion CPX v.1.0.2 [10], which combines CP and SAT techniques.

In Table 1 we show the cost of the best solution obtained by each solver within 5 minutes of running time. For our local search solver, we also report the average cost of 30 repetitions, obtained with the parameter configuration described in Section 5.2. The dash symbol states that the solver was not able to obtain any solution in the timeout.

The outcome is that the local search solver clearly outperforms all the other solution methods both on generated and real cases within the granted computational time. In detail, it was able to find the perfect solution for 11 instances; in addition in one case (gtt-real02) the perfect solution was obtained in all 30 trials.

6 Discussion and conclusions

We have modeled and solved a timetabling problem that, up to our knowledge, has not been formalized yet in the scientific literature. A similar problem has been considered in [7], in which however constraints and objectives are different; for

example, in their case the committee changes for each single student, so that the order of presentation is also important to minimize the time spent by committee members.

For this new problem, we have collected 3 real cases and developed an instance generator in order to abstain from overtuning on such a small number of available instances.

We have developed both a local search method and a MiniZinc model and tested several MiniZinc engines. The experimental analysis shows that the local search solver outperforms indisputably the MiniZinc engines, at least on the ground (instances and timeout) defined in the work. This is however not a fair comparison, given that the MiniZinc engines are exact solvers and they would need more time to explore their search tree effectively for instances of this size.

This model has been used to generate the graduation calendar for the Faculty of Psychology of the University of Milano-Bicocca and it has been used by the university with satisfactory results.

For the future, we plan to investigate on the management of the thesis defense procedure in other universities, in order to design a more general formulation. In addition, we plan to try to improve the MiniZinc model, with the aim of making it more suitable to be solved with the available engines.

Another future development is to collect more real-life instances, and to use them for the improvement of the generator, in such a way that it can create more realistic instances.

References

1. Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.
2. Mauro Birattari, Z. Yuan, P. Balaprakash, and Thomas Stützle. *F-Race and iterated F-race: An overview*. Springer, Berlin, 2010.
3. Luca Di Gaspero and Andrea Schaerf. EasyLocal++: An object-oriented framework for flexible design of local search algorithms. *Software—Practice and Experience*, 33(8):733–765, 2003.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.
5. John Michael Hammersley, David Christopher Handscomb, and George Weiss. Monte Carlo methods. *Physics today*, 18:55, 1965.
6. Jeffrey H. Kingston. Educational timetabling. In A. Sima Uyar, Ender Ozcan, and Neil Urquhart, editors, *Automated Scheduling and Planning*, volume 505 of *Studies in Computational Intelligence*, pages 91–108. Springer Berlin Heidelberg, 2013.
7. Beáta Kochaniková and Hana Rudová. Student scheduling for bachelor state examinations. In *Proc. of the 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA-13)*, pages 762–766, 2013.
8. Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.
9. Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming–CP 2007*, pages 529–543. Springer, 2007.
10. Opturion. Opturion CPX website. URL: `http://www.opturion.com/cpx.html`. Viewed: 2nd February 2015.
11. Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.
12. Christian Schulte, Mikael Lagerkvist, and Guido Tack. Gecode/Flatzinc website. URL: `http://www.gecode.org/flatzinc.html`. Viewed: 2nd February 2015.
13. Tommaso Urli. json2run: a tool for experiment design & analysis. *CoRR*, abs/1305.1112, 2013.