A Multineighbourhood Local Search Solver for the Timetabling Competition TTComp 2002

Luca Di Gaspero and Andrea Schaerf

Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica
Università di Udine
via delle Scienze 208, I-33100, Udine, Italy
email: l.digaspero@uniud.it schaerf@uniud.it

1 Introduction

The International Timetabling Competition 2002 has been organized by the Metaheuristic Network in order to compare different algorithms in a common measurable ground. All information about problem definition, instances, rules, and solution evaluation is available at the webpage `http://www.idsia.ch/Files/ttcomp2002/`, and for the sake of brevity we do not repeat them here.

We have participated in the competition, and in this abstract, we discuss our solution method, our scores, and we provide some personal comments about the general outcomes.

2 Algorithm description

Our solver is based on the local search paradigm. In particular, it relies on the multi-neighbourhood local search idea proposed in [1], which is based on the composition of neighbourhoods and the interleaving of different local search algorithms.

In details, our solver is composed by three stages that are interleaved sequentially, and the whole process is repeated until the total time is expired. Each stage starts from the final state reached from the previous one, and the first stage starts from a random state.

The search space is represented by two integer-valued vectors of size $|E|$ (where E is the set of events) that store, respectively, the timeslots and the rooms assigned to each event. The neighbourhoods employed are described below as modifications of the above vectors.

Stage 1: Hill climbing

The hill climbing procedure draws a random move at each iteration. The move is accepted if it improves or leaves unchanged the value of the cost function.

The neighbourhood used is the set-union of three neighbourhoods:

ChangeTime: changes the timeslot of an event (the room is left unchanged);

ChangeRoom: changes the room of an event (the timeslot is left unchanged);
SwapTime: swaps the timeslots of two events (each takes the room of the other).

The random selection is done in two steps: first we select the basic neighbourhood, with uniform probability (1/3 each), and then we uniformly draw a random move from the selected neighbourhood. Only non-trivial moves are generated and evaluated; that is, we avoid to move an event from a room r to the room r itself (and similar ones).

The stop criterion is based on the number of *idle iterations*, that is the iterations elapsed since the last improvement. We fixed this number to 300,000 for our solver.

The final state of this stage is the last one generated, which (for the hill climbing) has the minimum cost among the visited ones.

Stage 2: Tabu search

The tabu search procedure uses a variable-length short-term tabu mechanism (without any form of long-term prohibition). The neighbourhood used is:

ChangeBoth: moves an event in a different timeslot and a different room.

Not all the neighbours are evaluated, but only those that satisfy the following conditions:

- the event has a minimum number of students (set at value 3 in the solver);
- the target room is free in the target timeslot;
- the move is not tabu, that in our case means that it does not involve the same event of a move in the tabu list.

All moves that satisfy all three conditions are generated and evaluated for the selection of the best move at each iteration. The stopping condition of this stage is a compound one: the stage stops after a number of idle iterations that is inversely proportional to the density of the instance or when a fixed total number of iterations has elapsed. We allow less idle iteration to denser instances in order to allow roughly the same running times for each round. Regarding the number of iterations allowed, we grant the algorithm a total number of iterations equal to 3 times the number of idle iterations granted. The tabu list length varies from 20–30 to 30–40 according to the density of the instance. The final state of the stage is the one with best cost among the visited states.

Stage 3: Multi-swap shake

This stage performs only one single move, selecting the best neighbour from the following neighbourhood:

MultiSwap: all events in a timeslot are moved to a different timeslot, and vice versa. Rooms are kept unchanged.

The final state of this stage is the one obtained performing the best move, even if this is a worsening move. The purpose of this neighbourhood is to perturb the current solution, without introducing too many new violations.

Implementation

The algorithm has been implemented in the C++ language, compiled with `gcc` v. 3.2, and makes use of the EasyLocal++ framework [2] for the development of local search algorithms.

3 Results and discussion

For each one of the 20 instances we made 200 trials and we stored the best result. Our results have been verified by the competition organizers and are reported in the competition site along with the others.

Our solver resulted fourth in the general ranking, but it is actually the sixth best solver if we include also the two teams that where not included in the list because these were developed by members of the Network.

A detailed comparison of the solution methods is not easy and is still ongoing. However, at a first glance we believe that the main differences between our algorithm and the ones that ranked better in the competition reside in the following points.

Room assignment

Looking at the description of the solver classified ahead, it seems that all of them use the simplified search space as described in [4]. That is, the room assignment is not part of the search space but it is generated a posteriori by a matching algorithm that assigns events to available rooms, separately for each timeslot.

In retrospective, we see that probably our choice of using the larger search space, and not the matching was not the right one. In fact, the possibility of delegating the room assignment to the post-processor prevents the search to get stuck in states characterized by bad room assignments.

We followed such an approach in order to experiment a different line of research, which would have been entirely based on local search. This choice was dictated also by the possibility to extend the solver to more complex real settings, in which the matching is not a viable option. For example, if we are allowed to express a constraint requiring that events in different periods have to be scheduled in the same room, then the matching becomes more critical. In fact, the addition of constraints on the possible matching makes, in most cases, the matching problem NP-complete (see [3]).

Initial solution

All our competitors employ some more or less complex form of greedy/construction algorithm for building an initial feasible solution, whereas our algorithm starts from a randomly generated solution (possibly infeasible). Our experiments with good initial solutions are still under consideration, but so far they did not provide general improvements.

Neighbourhood exploration

Most of the algorithms proposed do not allow the search to move away from feasibility, whereas our algorithm deliberately retains the possibility to move from feasible to infeasible solutions. We decided to do it this way because we believe that in many cases the path leading from a good solution to a better one passes through some infeasible solutions. Unfortunately, though, the exploration of all infeasible neighborhood costs time, and it seems that for the competition instances the trade-off turned out to be in favour of the other choice. It might be the case, however, that in more difficult instances, i.e. instances with zero or a few feasible solutions, our choice could outperform the other.

Finally, differently from our algorithm, all the top-scoring procedures benefit of a "delta-evaluation" mechanism for speeding up the exploration of the neighbourhood. Such a mechanism consists in tabling the cost difference for all possible moves, instead of recomputing them at each iteration. The description of the winning algorithm reports that the delta-evaluation saves about 60% of the overall computational time. We plan to incorporate such a mechanism in our algorithm as well, allowing our procedure to perform more iterations in the same amount of time.

References

1. Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In Edmund Burke and Patrick De Causmaecker, editors, *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002)*, number 2740 in Lecture Notes in Computer Science, pages 262–275, Berlin-Heidelberg, 2003. Springer-Verlag.
2. 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.
3. Alon Itai, Michael Rodeh, and Steven L. Tanimoto. Some matching problems for bipartite graphs. Technical Report TR93, IBM Israel Scientific Center, Haifa, Israel, 1977.
4. Olivia Rossi-Doria et al. A comparison of the performance of different metaherustic on the timetabling problem. In Edmund Burke and Patrick De Causmaecker, editors, *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002)*, number 2740 in Lecture Notes in Computer Science, pages 329–351, Berlin-Heidelberg, 2003. Springer-Verlag.