

The Description of the Algorithm for International Timetabling Competition

Yuri Bykov

University of Nottingham, School of Computer Science & IT,
Wollaton Road, Nottingham NG8 1BB, UK
yxb@cs.nott.ac.uk

Submitted results are produced with the basic Great Deluge local search algorithm, launched for a predefined number of moves. The comprehensive investigation of the performance of Great Deluge algorithm can be found in our paper “A Time-Predefined Local Search Approach for Examination Timetabling Problems”, which is available on the web (<http://www.cs.nott.ac.uk/TR/2001/2001-6.pdf>). The main aspects of this algorithm (adapted to the course timetabling) are the following:

1. Data Structure

An input (static) data structure is represented as an array of records, each of them contains the information about a single event, such as: a number of students, available rooms, etc.

A dynamic data is organized for the purpose of minimization of processing time. Thus, several data structures are keeping in parallel:

- a) event/timeslots data;
- b) student/timeslots data;
- c) timeslot/events data;

Even the information is duplicated, this approach provides a quicker access to the necessary data.

2. Initialization

An initial solution is generated by modified Brelaz (saturation degree) sequential algorithm. An event with the minimum number of available timeslots is scheduled first. The modifications are as follows:

- a) The number of available timeslots is calculated while considering both clash-free requirements and room assignment.
- b) The algorithm provides a random feature: having a number of candidate events, the scheduled one is chosen randomly. The scheduled timeslot is also selected randomly.
- c) In the case, when there is no available timeslot for the scheduling of some event, the algorithm operates in the following way. A randomly chosen timeslot is completely rescheduled and the given event is scheduled into it. Then the procedure continues.

The produced initial solution is feasible, i.e. it is clash-free and all events can be assigned into rooms. After the initialization phase a local search phase is started.

3. Neighborhood

A new candidate solution is generated in the following way. The algorithm randomly chooses an event and a new timeslot for it. The candidate solution is accepted if it is feasible and satisfies an acceptance condition.

4. Feasibility

A feasibility is checked in two stages. Firstly - the clash-free requirement (no student has to sit two courses in the same time period). Secondly the room allocation requirement is checked. If the candidate solution is infeasible, the proposed move is rejected.

5. Great Deluge Acceptance Conditions

All feasible candidate solutions are checked whether they satisfy the Great Deluge acceptance conditions. Due to these conditions a candidate penalty should not be more than: a) the penalty of the current solution; b) the current value of “level”. The value of level is equal to the initial penalty at the beginning and is gradually reduced during the search. The rate of its reducing is calculated basing on the expected launch time (number of moves).

6. Launch Time.

The expected launch time is calculated in the following way:

- 1) An average processing time of one move is calculated (while performing 1 million of moves).
- 2) An expected number of moves is calculates in order to fit the algorithm into a specified time period.
- 3) The resulting number of moves is rounded by 5 millions in order to make the algorithm been independent on the small deviations in hardware speed.

7. Checking a Clash-Free Constraint

Algorithm simply checks all students (who have this event) whether the candidate timeslot is already occupied by somebody of them.

8. The Calculation of a Total Penalty

This procedure employs the fact that the number of variants of the distribution of events in a day (“day-maps”) for a single student is limited (in case of nine timeslots per day this number is 512). The penalties of all possible day-maps are pre-calculated in advance. Thus, the total penalty is just the sum of the penalties of all student’s day-maps for all days.

9. The Calculation of a Delta-Penalty

While evaluating any candidate solution the algorithm calculates only its delta-penalty (an amount, on which the penalty will be changed by the move). As the previous one, this procedure also employs a pre-calculated data. Here delta-penalties for all variants of removing/inserting an event from/into a day-map are known in advance. Thus, the total delta-penalty is just the sum of delta-penalties for all students who have the moved event. The “removing” delta-penalties are summarized for the old timeslot’s day, and the “inserting” delta-penalties – for the new day. Notice, that if the old and the new timeslots both belong to the same day, the procedure is slightly different. Here the “inserting” delta-penalties are chosen for the same day-map where the moved event is deleted.

10. Room Allocation

The room allocation procedure consists of two subroutines:

a) An algorithm, which iteratively evaluates a “room-event matrix” (the matrix (a_{ij}) of size $N_{event} \times N_{room}$, where $a=1$ means that the event i can be assigned into room j and $a=0$ means the opposite. This technique is the primary for the evaluating of the room-feasibility of candidate solutions.

b) A standard brute-force routine. This algorithm is applied in the cases where the primary technique fails and also for the final room assignment.

The primary technique (a) does not actually assign the rooms to events (as it is not necessary for intermediate solutions). It employs the proposition that a current timeslot’s room-event matrix is feasible and investigates whether the insertion of an extra row (correspondent to the inserted event) will keep the feasibility of the matrix.

The algorithm iteratively reduces the matrix and calculates the number of “unallocated” events while checking several particular cases, such as:

- if matrix contains an all-zero column in the position where inserted event contains 1 – the new matrix is feasible;
- if the number of remaining columns is less than the number of rows or the number of unallocated events – the new matrix is infeasible
- if the number of unallocated events is less than 5 – the new matrix is feasible. This assumption works only for proper reduced matrices (see below).

The reduction of the matrix is performed by iterative deleting of:

- all-zero columns and rows (the number of unallocated events is not reduced);
- columns, where inserted event contains 1 (the number of unallocated events is reduced);
- all-one rows (the number of unallocated events is reduced);
- rows and columns which contain single 1 and other zeros (the number of unallocated events is reduced).

The “iterative” means that after every reduction of the matrix its investigation starts from the beginning while not taking into account “deleted” rows and columns.

If the matrix cannot be reduced into any particular case, the brute-force algorithm (b) is applied. However the experiments showed that the primary algorithm “copes” with 99.5% of moves (matrices).

11 Stopping Condition

The Algorithm stops when no improvement is detected during 100000 moves.