

International Timetabling Competition 2002

David Bredström

March 30, 2003

The algorithm used is a local search method with different swap strategies. It has one basic algorithm (based on maximum cardinality matching) that takes a set of events from two slots and rematch them. It also uses an algorithm that takes three slots and tries to minimize the penalty by changing slots for the events.

Initially the algorithm puts the events in as few slots as possible, simply by making first one slot per event (always feasible) and second reduce the number of slots by moving events from one slot to another and removing a slot as soon as it's empty. This is repeated until no more moves are possible. Next the algorithm produces feasible solutions by repeating the following: Take two slots where at least one of them has one unused room and take one event. Do a tree search to find if the rooms can be reallocated so that all events fit in the two slots. This procedure gives feasible timetables (that is, less than 45 slots) for all instances.

Next follows the main loop. Given feasible solutions the tree search used for finding feasible solutions is the main move from neighbor to neighbor. It takes two slots and ranks the slots for every event, sorts the events starting with the most penalized event and puts them back in order. It accepts the first feasible combination found. If it's close to the best solution found, it applies exact (full tree search) 2-slot rematching and 3-slot heuristic rematching. To move away from local optima a randomness is introduced. It affects mainly the penalty dependent sorting in the tree search. The magnitude of the randomness is partly dependent on how hard the problem instance is, measured with the time it takes to find the initial feasible solution and how tight the instance is (for instance, if there are more features available in the rooms in average than needed by the the average event).

Note: The 2-slot rematching can be performed very fast since it starts with a feasible solution, that is there are never three events with the same student involved. This means that taken one event, all events with student clashes with that particular event, have to be put in the opposite slot. (can be seen as a "two color node coloring problem"). A maximum cardinality matching will then always produce feasible slots.

Computer used: P4 1700 MHz, 1 GB memory running Linux RedHat 7. Benchmark results:

This program took 89 seconds to run on your machine

This means that you can run your algorithm on the instances provided for 534 seconds