The Second International Timetabling Competition, A depth-first search approach

Onno Wink

Independent, Denver Colorado, USA, onno.wink@uchsc.edu

Abstract. This abstract gives a concise description of the method that was used in an attempt to 'solve' the 16 problems as part of the second international timetabling competition.

Keywords: depth-First search, scheduling, heuristics

1 Description of the method

A randomly generated initial instance of the schedule will act as the root node of a depth-first search. Two lists of nodes will be kept (done and open). In every instance the best node from the open list is chosen, put in the done list and used to generate a set of child nodes. Every newly generated child node that corresponds to a schedule instance that is not in any of the lists is put in the open list. This process is continued until an improvement is found, in which case all lists are cleared and the search starts over again. In the following a more detailed description of the individual steps is given:

1) Create root node based on current instance
2) Loop over all events and for each violation (first hard then soft) generate a node for all other valid time slots that this event is allowed to move. For every such move compute the number of *affected* events, meaning, events that are now causing a violation because of this move, and put these in the open list.
3) Pick the best node from the open list and put it in the done list.
4) For every affected event (that is not yet in the move), create a new set of child nodes if the corresponding schedule did not already exist (in done or open list) and continue with 3. If any child corresponds to an improvement with respect to the root node continue with 1.

The search space (and the number of redundant nodes) is drastically reduced by only focusing on the set of affected events caused by the moves alone. In addition whenever an event is affected that already has been part of a move, the corresponding node will not be put in the open list. In another attempt to reduce the search space, the number of affected events plus the amount of moves should be within a set bound. In the actual implementation nodes are already popped from the open list in order to find an improvement while the events are traversed to see whether they are causing a violation.