The Second International Timetabling Competition, Track 3: Description of the Beam Search Based Algorithm

Donatas Elvikis*

Department of Mathematics, University of Kaiserslautern,
Paul-Ehrlich-Str. 14/461, 67653 Kaiserslautern, Germany
{elvikis}@mathematik.uni-kl.de
http://optimierung.mathematik.uni-kl.de

Abstract. We describe solution method applied to solve timetabling problems presented in the Track 3 of ITC2007. Heuristic approach is divided in two major parts: hard constraint solver and soft costs minimizer. Feasible solution is constructed by simple graph coloring and local search methods and afterwards heuristic, based on the beam search algorithm, is applied to minimize the soft costs.

Key words: Educational Timetabling, Heuristics, Beam Search

1 Introduction

Idea of solution algorithm came while analyzing *beam search* method [1], which is actually a bridge between *breadth-first* search and *best-first* search heuristics. Namely, it builds a solution tree by expanding nodes layer by layer, however conversely to breadth-first algorithms it only continues expanding w most promising nodes. This allows to avoid memory consumption problems without paying the price of inadmissibility pruning nodes which could lead to an optimal solution. Recently variations of beam search were successfully applied for solving STRIPS planning (see Zhou and Hansen [6, 7]) and scheduling (see Morton [5], Croce and T'kind [3]) problems. Here we describe how beam-stack search [7] was adapted for the local search procedure in the timetabling framework.

2 Algorithm Description

Description of the algorithm is given in this chapter. Solution ranking rules defined by the competition organizers clearly imply *lexicographical objective* function $f = Lex(f_h, f_s)$, where f_h stands for *hard constraint* violations and f_s — *soft*

* The author thankfully acknowledge partial support from the Deutsche Forschungsgemeinschaft (DFG) grant HA 1737/7 "Algorithmik großer und komplexer Netzwerke", New Zealand's Julius von Haast award and the Rheinland-Pfalz cluster of excellence "Dependable adaptive systems and mathematical modeling".

costs. Hence the first goal is to reach feasibility while minimizing soft constraint violations stays secondary aim. Following the landmarks set by organizers algorithm is divided into two major parts: namely, hard and soft constraint solvers which are partially interrelated.

2.1 Data Handling

The data container classes were inherited from the solution validation source provided by the Track 3 organizers. Additionally `Faculty` class was supplemented with a list of all *lectures*, i.e. $\mathcal{L} = \{l_1, \ldots, l_L\}$, that have to be scheduled to the *slots* in $\mathcal{S} = \{s = (p, r) \mid p \in \mathcal{P}, r \in \mathcal{R}\}$. Here lecture is defined as a part of *course* $c \in \mathcal{C}$ and slot is an element of the Cartesian product of *periods* $\mathcal{P}$ and *rooms* $\mathcal{R}$. Moreover we keep a conflict graph $G = (V, E)$, where V and E are sets of vertices and edges, respectively. All lectures and periods are associated with vertices $v_i \in V$. Edges are added to the graph G if either lectures l_i and l_j in $\mathcal{L}$ associated with $v_i, v_j \in V$ are conflicting or period $p \in \mathcal{P}$ is not available for lecture $l \in \mathcal{L}$. Collision graph is implemented as an adjacency list data structure which is provided in the BGL[1] library. G is used as a helper data structure for faster access to conflicting lectures and forbidden periods.

2.2 Solving Hard Constraints

Hard constraint solver consists of two components: graph coloring and local search. Simple greedy type heuristic implemented in BGL is used for coloring graph vertices. First all vertices are sorted by *smallest-last* ordering [4] and then *sequentially colored* [2] with respect to the given order. Note that there may be more lectures assigned to some period $p \in \mathcal{P}$ than rooms available in p. Moreover, lectures without assignment to any period may appear, i.e. there exists lecture $l \in \mathcal{L}$ with some color k such that there is no period p having color k. In the first case algorithm greedily assigns lectures to the rooms using *best-fit* ordering, i.e. we assign lecture to the smallest available room r while minimizing room capacity constraint violations. Lectures without assignments as well as lectures without common color with periods are left for further considerations. Algorithm is iteratively applied until no lecture l can be assigned to some slot 1. This way constructed timetable has no conflicting lectures in the current assignments, however it may have a number of unscheduled lectures.

Note that sequential coloring algorithm depends not only on the supplied ordering but also on the adjacency structure of the graph G, i.e. in which adjacency order vertices are connected. In order to avoid systematically bad coloring solutions, depending on the order of suplied problem data, conflict graph is constructed in random order, namely the order of courses is shuffled and then G is build.

Almost all instances could be solved to feasibility using simple coloring heuristic, however for instances 1, 2, 7 and 10 feasibility could not be reached in none of

[1] BGL – Boost Graph Library: `http://www.boost.org`

the test runs. Hence additional local search procedure was added with maximum runtime limitation of 1 second. Local search procedure contains two operations: *lecture shifting* and *kick off*.

Shift operation randomly chooses unscheduled lecture $l \in \mathcal{L}$ and tries to find slot $s \in \mathcal{S}$ such that either it is empty or lecture l' scheduled in s could be moved to some other free slot $s' \in \mathcal{S}$ without causing any conflicts. If it does not succeed, then algorithm looks for a slot s such that exchanging lectures l and l' in s does not cause additional f_h costs. However if predefined number of steps do not improve the best known schedule, then kick off operation is performed.

Kick off randomly chooses unscheduled lecture $l \in \mathcal{L}$ and some feasible slots $s \in \mathcal{S}$ for l. Here by feasibility of slot is meant that s in period p is not forbidden for the course $c \in \mathcal{C}$, where $l \in c$, and no other lecture l' in course c have an assignment to the same period p. If such lecture and slot exists, then l is assigned to the slot s and all conflicting lectures are kicked off of their current assignments.

Note that kick off operation is done on the best currently known solution, whereas lecture shift starts with the best know and continues iterating on the last generated timetable, even if f costs gets worse than those of the best schedule.

2.3 Solving Soft Constraints

Soft constraint solver is based on the beam search, namely an adaption of the beam-stack search algorithm [7] for the timetabling problem. Zhou and Hansen improved classical beam search method with systematic backtracking, hence enabling algorithm quickly find initial solution and further continue search for an optimal until one is found or allowed time is exhausted.

Algorithm uses the global *upper bound* UB and a *stack* with lower and upper bounds on f for each layer, denoted by f_{min} and f_{max} respectively. Costs bounds equal to $f_{min} = (0, 0)$ and $f_{max} = (UB_h, UB_s)$ are pushed to the stack each time a new layer is expanded. Solution tree node is pruned whenether its costs f satisfies $f >= UB$ or $f < f_{min}$. If layer contains more nodes than the *beam width* w, then only the most promising nodes are kept for further consideration and f_{max} is set to the minimum costs of the rejected or pruned nodes. Note that rejected nodes are not pruned as they still have a chance to be expanded after the *backtracking* step. In order to track pruned solution nodes algorithm uses caching system.

Backtracking is done whenether an *empty layer* is found, i.e. layer where all nodes are pruned by the rule presented above or nodes are leaves of the solution tree. Algorithm backtracks until the layer with $f_{max} < UB$ is found. Every time algorithm backtracks it shifts the layer costs bounds to the right, namely $[f_{min}, f_{max}) = [f_{max}, UB)$ in such way systematically forcing the search beam to admit a different set of successor nodes and thus continuing search for an optimal solution.

In spite of improvements introduced by Zhou and Hansen [7] beam-stack search is far too slow in the framework of competition, hence we use it only as a local search heuristic to explore solution space. Instead of starting to construct

solution from the first layer, i.e. an empty schedule, we start directly by backtracking from the solution found after applying hard constraint solver. In the solution tree each layer corresponds to the number of assignments of lectures to slots in the timetable, hence all assignments in the schedule are kept in the order they were added. Whenever algorithm backtracks, it removes the last assignment from the timetable. This way algorithm performs local search by trying to reschedule lastly assigned lectures.

In order to limit the number of steps performed on one schedule we set two parameters: execution time t_{max} and maximum number of backtracks b_{max}. Whenever beam search violates one of the given parameters, the order of assignments in the current best timetable is shuffled and beam search is repeated, thus local search is done in the other region of solution space. However given bounds might be too tight, therefore we increase t_{max} and b_{max} values stepwise using the following pattern which depends on the number of b_{max} violations without improving timetable costs:

#Backtracks	t_{max}, (s)	b_{max}
0	1	3
300	1	4
900	2	5
every 3000	+2	+1

Initially we set t_{max} and b_{max} to 1 second and 3, respectively. If no improvements can be found in the local solution search space we increase the values, hence solution search space is broadened. Whenether an improved solution is found, #Backtracks is set to 0 and algorithm continues by first examining small solution space of the current schedule.

References

1. Bisiani, R.: Beam Search. In S. Shapiro, editor, Encyclopedia of Articial Intelligence, Wiley & Sons, pp. 56–58 (1987)
2. Coleman, T. F., More, J. J.: Estimation of sparse Jacobian matrices and graph coloring problems. Journal of Numerical Analysis, vol. 20, pp. 187–209 (1983)
3. Della Croce, F., T'kindt, V.: A Recovering Beam Search algorithm for the one-machine dynamic total completion time scheduling problem. Journal of the Operational Research Society, vol. 53, pp. 1275–1280 (2002)
4. Matula, D., Marble, G., Isaacson, J.: Graph coloring algorithms in Graph Theory and Computing. Academic Press, pp. 104–122 (1972)
5. Ow, P. S., Morton, T. E.: Filtered beam search in scheduling. International Journal of Production Research, vol. 26, pp. 35–62 (1988)
6. Zhou, R., Hansen, E.: Breadth-First Heuristic Search. 14th International Conference on Automated Planning and Scheduling (2004)
7. Zhou, R., Hansen, E.: Beam-Stack Search: Integrating Backtracking with Beam Search. 15th International Conference on Automated Planning and Scheduling (2005)