High School Timetabling: Modeling and solving a large number of cases in Denmark

Matias Sørensen · Thomas R. Stidsen

Keywords High School Timetabling · Modeling · Adaptive Large Neighborhood Search

1 Introduction

A general model for the timetabling problem of high schools in Denmark is introduced, as seen from the perspective of the commercial system Lectio¹, and an Adaptive Large Neighborhood Search (ALNS) algorithm is proposed for producing solutions. Lectio is a general-purpose cloud-based system for high school administration (available only for Danish high schools), which includes an embedded application for creating a weekly timetable. Currently, 230 high schools are customers of Lectio, and 191 have bought access to the timetabling software. This constitutes the majority of high schools in Denmark.

This large customer base entails a need for a model of the problem which is general enough to suit many different requirements, while still remain tractable by computer aided solution methods. This supports the recent trend of developing general models for timetabling problems (see Burke et al (1998); Asratian and de Werra (2002); Özcan (2005); Causmaecker and Berghe (2010); Bonatti et al (2010); Post et al (2011, 2012)). Furthermore, the timetabling problem of Danish high schools has not been formally described in the literature before. Some recent formulations of related problems from other countries include Wright (1996); Wood and Whitaker (1998); Bufo et al (2001); Melicio et al (2005); Avella et al (2007); Nurmi and Kyngas (2007); Boland et al (2008); Santos et al (2010); Minh et al (2010).

¹ http://www.lectio.dk [lectio@macom.dk], developed by MaCom A/S, Vesterbrogade 48 1., DK-1620 Copenhagen V.
2 Model formulation

The following sets are given:

- **Timeslots**, usually made up of 5 days and 4-8 daily timeslots.
- **Entities**, the combined set of students and teachers.
- **Classes**, a subset of entities which is taught/teaching a specific topic.
- **Rooms**
- **Events** (corresponding to lectures), the basic timetabling-unit which must be assigned exactly one timeslot in the timetable.

For each class, a number of events is given (usually between 2 and 5). The basic timetabling problem concerns the assigning of each event to a timeslot, and a room to each event, such that no clashes among entities or rooms occur.

Furthermore we introduce the concept of EventChains, which separates this problem from related problems described in the literature. An EventChain consists of a subset of events, and each of these events are assigned an offset. At least one event must have offset 0, corresponding to the start of the EventChain. All events of an EventChain with the same offset must be placed in the same timeslot. All events of offset 1 must be assigned the timeslot following the timeslot assigned to events of offset 0, and so forth. See also Fig. 1. No restrictions are posed on which events can be included in the same EventChain, and the offsets of events in an EventChain are completely for the user to decide, providing a lot of flexibility. E.g. a double lecture can be set up by creating an EventChain consisting of two events for the same class, with offsets 0 and 1, respectively. EventChains also allows for parallel double lectures for several classes, triple lectures, grouping of elective classes in the same timeslot, etc. Many of such features have been requested by the users of Lectio, and EventChains are a pretty generic way of solving them.

**Fig. 1** Six EventChains placed in a partial timetable.

Fig. 2 illustrates the data-model by an example with two classes, each assigned three events, and two EventChains.

Besides the no-clashes constraints, the following hard constraints are included in the model:

- An event cannot be assigned to any of its forbidden timeslots
- Each event must be assigned an admissible room
Fig. 2 Two classes and their respective events, and two EventChains.

- Only one event of each class per day (unless otherwise specified in the given EventChain)
- For each teacher, a required number of days-off is given

The following weighted objectives (soft constraints) are used:

- Maximize the number of events assigned to a timeslot (very high priority)
- Maximize the number of events which are assigned a room (high priority)
- Maximize the number of days-off for teachers (low priority)
- Minimize idle slots for entities (medium priority)
- Minimize the amount of different rooms which are assigned to events of the same class (room stability constraint, low priority)
- Minimize the number of occurrences of two events of the same class being assigned two consecutive days (neighbor-day constraint, low priority)

3 Adaptive Large Neighborhood Search

ALNS is a recent extension of the Large Neighborhood Search (LNS) paradigm, often credited to Ropke and Pisinger (2006). As in the LNS framework, first a destruct (ruin/remove) operator is applied to the solution at hand, and then a construct (recreate/insert) operator is used to repair the solution. In an ALNS framework, multiple destruct and construct operators are used, and the adaptive layer keeps track of their individual performance, and increases the probability of selecting operators which have previously performed ‘good’. ALNS has mainly been applied to variants of the Vehicle Routing Problem (VRP) (Aziz et al (2010); Hemmelmaier et al (2011); Salazar-Aguilar et al (2011); Ribeiro and Laporte (2012)), but lately also other problem-domains (Müller et al (2011); Muller (2010); Kristiansen et al (2011); Kristiansen and Stidsen (2012); Sørensen et al (2012))

We propose here an ALNS heuristic for solving the described timetabling problem. The following remove- and insertion-operators are used:

- InsertGreedy: Insert EventChains in greedy way based on contribution to objective. At each insertion, also attempt to assign rooms to inserted events.
• InsertRegretN: This is similar to the Regret-N neighborhood applied to variants of the VRP (Tillman and Cain (1972); Martello and Toth (1981); Potvin and Rousseau (1993)). The regret-measure is specified in terms of best and second-best insert-move for a given EventChain.

• RemoveRandom: Select N random EventChains and unassign them.

• RemoveRelated: Related to Shaw operator (Shaw (1997, 1998)). Select EventChains to remove based on their similarity between classes, feasible rooms and entities.

• RemoveTime: Select a random timeslot, and remove EventChains assigned to it. Repeat until N EventChains has been removed.

• RemoveClass: Select a random class, and remove EventChains which contains it. Repeat until N EventChains has been removed.

Furthermore we apply a special set of operators, RoomRemove/Insert, which are coupled. In a coupled set of operators, the choice of remove operator implies the choice of repair operator. In RoomRemove, N random room-assignment are removed from the solution. In RoomInsert, rooms are assigned to events in a greedy way.

4 Preliminary results

The Lectio database contains about 4000 datasets from 94 different high schools. Grouping these by school and year entails about 200 'unique' datasets. The authors plan to make at least some of these datasets public, most likely using the XHSTT format (Post et al (2012)).

Table 1 shows statistics and preliminary computational results for six datasets. These results show that the heuristic finds solutions where many events are unassigned, but if they are assigned a timeslot, a suitable room is also assigned. In the full paper, comprehensive computational studies will be made. This will include comparison of the found solutions with a bound provided by an integer programming model.

<table>
<thead>
<tr>
<th>Name</th>
<th>#E.</th>
<th>#E.C.</th>
<th>#R.</th>
<th>#C.</th>
<th>#Ent.</th>
<th>#T.</th>
<th>#E./w pos.</th>
<th>#E./w room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorø2011</td>
<td>424</td>
<td>424</td>
<td>70</td>
<td>132</td>
<td>230</td>
<td>50</td>
<td>360.6 (13.6)</td>
<td>360.6 (13.6)</td>
</tr>
<tr>
<td>Skive2010</td>
<td>2955</td>
<td>1749</td>
<td>58</td>
<td>331</td>
<td>304</td>
<td>90</td>
<td>2358.2 (13.9)</td>
<td>2358.2 (13.9)</td>
</tr>
<tr>
<td>Fjerritslev2009</td>
<td>530</td>
<td>387</td>
<td>122</td>
<td>40</td>
<td>487.9 (2.9)</td>
<td>487.9 (2.9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ViborgT2010</td>
<td>536</td>
<td>447</td>
<td>21</td>
<td>136</td>
<td>85</td>
<td>50</td>
<td>515.2 (1.8)</td>
<td>515.2 (1.8)</td>
</tr>
<tr>
<td>Herning2010</td>
<td>1671</td>
<td>1616</td>
<td>86</td>
<td>355</td>
<td>213</td>
<td>60</td>
<td>1545.2 (2.5)</td>
<td>1545.2 (2.5)</td>
</tr>
<tr>
<td>Hæsser2011</td>
<td>1843</td>
<td>1080</td>
<td>79</td>
<td>404</td>
<td>578</td>
<td>50</td>
<td>1331.8 (3.2)</td>
<td>1331.8 (3.2)</td>
</tr>
</tbody>
</table>

Table 1 Preliminary results for 10 runs of ALNS heuristic on 6 datasets. Columns 2-9 shows the number of Events, EventChains, Rooms, Classes, Entities, Timeslots, Avg. events assigned to a timeslot, Avg. events assigned to a room, respectively. For columns 8 and 9, also the standard deviation is shown.
References


