Cross-Curriculum Scheduling with THEMIS
A Course-Timetabling System for Lectures and Sub-Events

Abstract We report on a practical implementation of a curriculum-based course-timetabling system for pre-enrolment scheduling that is successfully used in our university. The implementation is based on a sophisticated model that captures essential real-world requirements in terms of course-structure modelling. Our tool THEMIS allows to handle courses that have sub-events and that are shared between different programs of study. It can also consider whether a shared course is mandatory or optional in each curriculum. THEMIS supports a cyclic and interactive workflow and offers comfortable means for editing model data and timetables.

Keywords system demonstration course-timetabling system real-world model practical implementation cross-curriculum scheduling sub-events

1 Specific Challenges in Real-World Course-Timetabling

Various constraints of different type, uncertain information and competing goals turn curriculum-based course timetabling for real-world settings into a challenging task [2]. In case of our department we observe that many aspects of this scheduling problem can be modeled using typical entities, constraints and cost components. In particular, courses are attended by students from different programs of study and each program has its own curriculum. E.g., the course *Theoretical Computer Science* has first-year students from the two Bachelor programs *Computer Science (CS)* and *Internet-based Systems* (IBS), and second-year students from the Bachelor program *Digital Media and Games* (DMG). For a standard model that covers most aspects of our setting we refer to CB-CTT from [1]. However, to obtain practical solutions we also need to consider some additional requirements:

– Each course has not only lecture events but also a number of smaller sub-events associated with it, like tutorials or laboratory classes. All students attending a lecture are partitioned into these sub-events, each of limited size.

Heinz Schmitz (corresponding author), Christian Heimfarth
Trier University of Applied Sciences
Department of Computer Science, Schneidershof, 54293 Trier, Germany E-mail: schmitz@informatik.fh-trier.de

- The same course can be mandatory for some students but optional for others, depending on their program of study. E.g., students from IBS have to take the course *Web-Technologies*, while CS students may choose this or some other course to fill one of the placeholders in their curriculum. Collisions between optional courses should be avoided to offer students a large number of possible choices.
- Lectures and sub-events can require more than one timeslot. In some cases, even sub-events of the same course have different numbers of timeslots to account for different skill levels.

As a consequence, we have strong dependencies in terms of clashing constraints across different curricula. Moreover, we need to construct a timetable for each term *prior* to student enrolment. So there is only limited information about what students from which program attend what lectures, and we have no information about sub-event enrolments. Also, several other organisational requirements have to be taken into account: No disruption or noticeable re-scheduling during a period is wanted, and, on the other hand, there is strong need for manual editing and updating, especially during the first weeks. The typical quantity structure of a problem instance for our department has about 800 students, 25 teachers and 140 events to be scheduled in 27 timeslots and 15 rooms. We must consider curricula of three Bachelor programs, two Master programs and some other post-graduate training programs, and we expect that the number of programs and events increases in the next years. Altogether, we are faced with a complex scheduling problem for which it seems nearly impossible to obtain feasible or even optimized solutions without strong tool support.

2 Overview of THEMIS

The ambitious goal of THEMIS is not only to implement some experimental algorithms but to provide a reliable and comfortable software system for our schedulers that *really* solves the *real* problem. In this sense THEMIS can be understood as a contribution to the research agenda set up by McCollum in his paper [2]. We started THEMIS in 2006 and it is under continuous develoment since then, including a complete re-implementation in 2009 to account for the lessons learned. Right now THEMIS is successfully used to produce workable and optimized timetables in our department and in other departments of our university.

Inspired by the manual work of our schedulers prior to THEMIS, the tool supports an interactive and cyclic workflow consisting of the steps (1) management of model data, (2) allocation of anonymous groups of students to lectures and sub-events, (3) automatic timetable generation, (4) manual editing of timetables, (5) presetting of (parts of) a timetable, and returning to (1), (2) or (3).

2.1 Model Data (Step 1)

An independent set of model data, usually one per scheduling period and institutional unit, is organized in a *project*, typically called (*CS-Department*, *SummerTerm2010*), (*EngineeringDepartment*, *WinterTerm2009*) and so on. This structure allows to model different scenarios for the same period independently. The user can copy existing projects and reuse model data.

Information Modelling. THEMIS allows to handle the typical main entities in course timetabling, as there are timeslots, lectures and their sub-events, teachers and rooms, all with a number of specific attributes and relations among each other. For example, a course has a lecture event and a number of sub-events of a maximal size; an event requires one or more timeslots, has one or more teachers and requires or excludes a number of resources offered by rooms (e.g. computer workstations); teachers have, among other attributes, *preferred*, *available* and *not available* timeslots, and so on. Moreover, we have introduced an entity called *curriculum-semester-combination* (CSC) to model groups of students that need to follow a certain set of lectures determined by the curriculum they are enrolled to. Typical values are *BachelorCSSecondSemester* or *BachelorIBSFifthSemester* and the like. We determine the number of students in each CSC and assign one or more CSCs to each course to express its multiple usage in different curricula.

Solution Modelling. As usual, we distinguish between *hard constraints* that a timetable must fulfill to be feasible, and *soft constraints* that it should additionally fulfill in order to be 'good'. Right now, our list of constraints includes typical hard constraints, like 'no two events in the same room at the same time' or 'if two events are modelled in *mustFollowTo*-relation, then the timeslot of the second event must immediately follow the timeslot of the first event on the same day'. THEMIS also knows a rather large number of soft constraints that can be used to optimize feasible timetables. Each violation leads to penalty points that are accumulated for a timetable. Examples are 'minimize free timeslots between events for students of the same CSC', 'use preferred timeslots of teachers' and 'a sub-event should not be the only event on a day for a CSC'. Clearly, accumulating penalty points blurs the boundaries between different optimization objectives. So it is important to visualize for the user how the sum of penalties of a timetable is composed. THEMIS offers a *tree view* that clearly presents all details of a timetable score. Moreover, the user can choose weights to assess different objectives, up to the possibility to exclude objectives from optimization by choosing weight zero.

Each project memorizes the list of all timetables that have been generated so far in this project, so it is always possible to go back to earlier attempts. Each run of a generating algorithm adds a new timetable to this list. All timetables in a project are dynamically evaluated with respect to the current set of model data, i.e., in case of an update, all timetables in the project are automatically re-evaluated to determine feasibility and penalties.

2.2 Lectures and Sub-Events (Step 2)

Lectures and sub-events together with their associated mandatory and optional CSCs impose extra complexity to timetable scheduling. We briefly describe our approach to this problem with help of an idealized and reduced example.

Example. Suppose the course *Web-Technologies is* mandatory for the CSC *BachelorIBSThirdSemester* with 50 students and optional for the CSC *BachelorCSFifthSemester* also with 50 students. We estimate from past terms that 25 students from *BachelorCSFifthSemester will* choose this course and introduce four sub-events for it, each with a limit of 20 students.

After these steps are carried out for all courses in the project, we partition each CSC in anonymous blocks of students and map these blocks to the sub-events of the courses.

This is sufficient if a course is mandatory for a CSC since all of its students attend the lecture. In case of an optional course for a CSC, we partition only the estimated number of attending students into such blocks and map them to the sub-events of this course as well.

Example. (continued) The number of 50 students from *BachelorIBSThirdSemester* is partitioned into blocks of 20, 20 and 10, the number of 25 students from *BachelorCS-FifthSemester* into blocks of 10, 10 and 5. As a result we get two sub-events of *Web-Technologies* each with 20 students from IBS, one sub-event with 15 students from CS only, and one mixed sub-event of 20 students.

THEMIS has algorithms that support these partitioning and mapping steps. To partition the number of students it chooses the smallest possible number of blocks, each having a size from a user-defined range. These blocks are the basic units to allocate students to sub-events. The mapping is done by a greedy algorithm that assigns blocks to sub-events such that the heterogeneity with respect to different CSCs is minimized. The algorithm considers blocks in descending order w.r.t. their size and assigns them to sub-events in a best-fit manner. We do so because we expect less clashing conflicts during timetabling if CSCs share only few events. Partitioning and mapping is carried out as a preliminary step before timetable generation as part of the model data. Clearly, this partitioning and mapping step is a non-trivial optimisation problem on its own that deserves further investigation. Blocksizes and mappings can also be edited manually during the overall interactive workflow.

After this step we have information in our model about what students from which CSC attend what lectures and sub-events. This is further exploited to determine clashings of events during timetable generation by specific hard and soft constraints. An example of such a hard constraint is 'no two sub-events of two mandatory courses in a CSC with the same associated block in the same timeslot'. Penalties result, e.g., from 'two sub-events of different optional courses of a CSC in the same timeslot'.

2.3 Timetable Generation and Editing (Steps 3, 4 and 5)

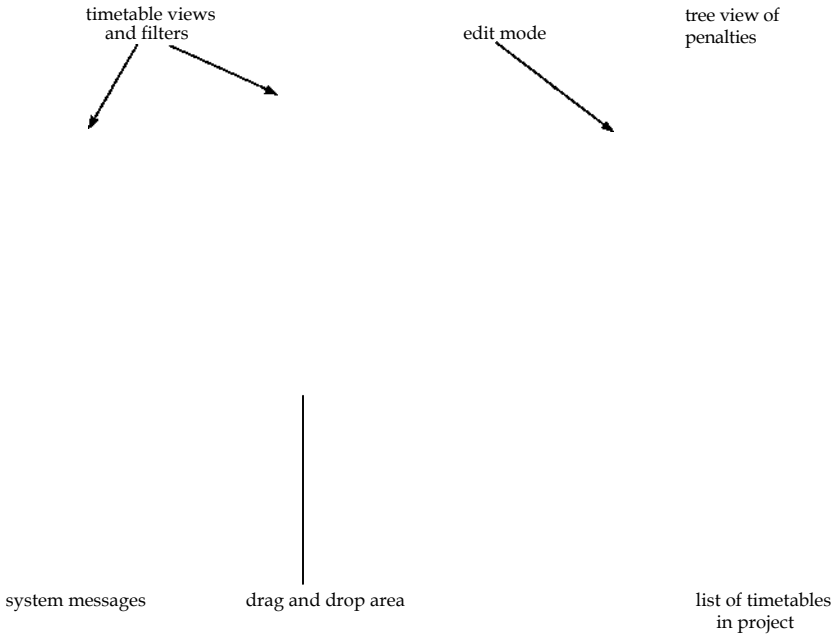
The user can choose to call an algorithm from scratch or to select any existing timetable in the project as an initial solution for some timetable-generating or improving algorithm, respectively. So far we have implemented the following set of algorithms (for common algorithmic approaches to this kind of problem see, e.g., [3]):

1. A constraint-based algorithm to obtain a feasible timetable (an efficient implementation of backtracking with forward-checking, degree heuristic, minimum-remaining-value heuristic and least-constraining-value heuristic).
2. A variant of algorithm 1 where the order in which the (timeslot, room)-values for each event are chosen depends on the penalty of the resulting partial timetable.
3. A local-search procedure with various parameters to improve feasible timetables.

All algorithms display their current best values and can be interrupted by the user. Algorithms 1 and 2 are based on a careful analysis of all hard constraints to reduce the range for the variables in advance and during backtracking. It turned out that in particular algorithm 1 is useful to reveal inconsistencies in model data very early. While the resulting penalty after algorithm 1 is fairly high, we obtain optimized timetables with small penalties from algorithms 2 and 3.

THEMIS has comfortable drag&drop-support for editing timetables. In the *free mode* events can be moved arbitrarily to any timeslot. However, schedulers find it very helpful to work with the *supported mode* of THEMIS during timetable editing. After choosing an event all timeslots are coloured red or green, depending on whether a move of this event to that timeslot results in a feasible timetable or not. Moreover, when dragging over red timeslots, the user is provided with information about what constraints are violated. In case of a green timeslot the new penalty is displayed in advance. Changing the room of an event is assisted by a similar mechanism in this *supported mode*. It is also possible to manually delete and insert events into an existing timetable.

Also other features of THEMIS turned out to be useful in practice. To display only specific aspects of a timetable it is possible to use filters, e.g., to show the timetable for a certain CSC, a certain teacher or a certain room. Timetables can be exported in a universal format for further publishing. Moreover, in order to support an incremental approach THEMIS allows the user to freeze parts of a timetable. As a consequence, all algorithms must maintain this presetting. Schedulers use this feature to produce similar timetables when single entities are added or updated. The following figure gives an impression of the screen for editing timetables.



2.4 Software Architecture and Engineering Aspects

The current release of THEMIS is realized as a pure java application based on the frameworks Hibernate[1] and Docking Frames[2]. It has a modular architecture with separate components for algorithms, graphical user interface and data management. Deployment

1 http://www.hibernate.org
2 http://dock.javaforge.com

is rather easy since THEMIS comes as a single jar-file, already including its database HSQLDB[3] (which can easily be changed to any other database working with Hibernate). We want to point out some critical aspects that we have paid attention to while developing THEMIS, but which do not deal with algorithm design in particular:

- Special care must be taken to maintain system-wide *data consistency*, i.e., due to complex dependencies between model entities, referential integrity must be carefully controlled when edit and update actions are preformed. This also includes some thoughts on storage management for the persistent entities in the model.
- There are parts in the code that are frequently executed and where the user expects very fast response times. Among others, *efficient implementations* of feasibility checks are needed. This is usually carried out on the data-structure level and cannot be discovered in some abstract pseudo-code from a research paper.
- Common *software-engineering principles* like design-patterns, encapsulation and no-duplicate-code must be strictly followed. Especially model entities and code to check constraints tend to spread all over the source code with the consequence, that maintenance and further development of the system become impossible.
- It is helpful to work with a single programming environment and language which leads to *seamless debugging* of the complete application. The latter is often problematic when different programming languages are used at the same time. We found that algorithms can be implemented in java reasonably fast (compared to other languages) when restricted to native data types.
- Due to the complex nature of the application domain there is strong need for *quality assurance* in the development project.

From our experience, disregarding a single of these aspects can make the difference between a working system and an instable prototype which cannot be used in practice. As a consequence, there is need for various expertises in the development team which makes such a project attractive also from an educational point of view (for Computer Science students). Luckily, we observe a high motivation of students to contribute to a system that affects their own academic calendar.

3 Future Work

THEMIS is primarily designed and used to produce timetables for single departments in universities. This is an appropriate approach in our case since only a small number of rooms is centrally owned and must be shared between different departments. However, we observe that there is an increasing number of programs in preparation that are offered in cooperation between two or more departments which implies that a common timetable is needed. We will investigate how THEMIS behaves on these larger instances and what new requirements arise.

Moreover, we want to further investigate algorithmic and modelling aspects of cross-curriculum scheduling of mandatory and optional courses and their sub-events (section 2.2). One aspect is that it seems to be difficult to generate timetables that guarantee a certain minimal number of non-overlapping optional lectures and sub-events in each CSC.

3 http://hsqldb.org

Another aspect is the presence of uncertainty in the input data which cannot be avoided in pre-enrolment scheduling. This becomes even more problematic if students from different programs need to be allocated to common sub-events of optional courses. THEMIS in its current version offers means to control this allocation in terms of block mappings as described above. However, the critical part in this approach is that it is based on the educated guess of the scheduler about how many students of a CSC will presumably attend what optional courses. In practice, this uncertainty is now handeled by scheduling small over-capacities (an approach with definitely tight limits) and manual editing after enrolment, e.g., by introducing additional sub-events. We think that there is strong pratical motivation to further investigate models and algorithms for timetables that are *robust* with respect to this uncertainty. A first step could be to identify and select scenarios that represent situations where students have chosen other optional courses than expected. Timetables should then be evaluated with respect to their ability to remain feasible in different scenarios.

Acknowledgements We are grateful to the anonymous referees for their helpful comments. We would also like to thank all other current and former members of the THEMIS development team for their contributions which are F. Hermes, P. Kranz, J. Pauken, P. Schiffgens, B. Schumacher, J. Sonntag, S. Stoffel, M. Stüber, M. Weiser. We are thankful to the Nikolaus-Koch-Foundation for their financial support.

References

1. F. De Cesco, L. Di Gaspero, and A. Schaerf, *Benchmarking Curriculum-Based Course Timetabling: Formulations, Data Formats, Instances, Validation, and Results,* In: E. Burke and M. Gendreau (eds.), Proceedings of PATAT '08 (2008).
2. B. McCollum, *University Timetabling: Bridging the Gap between Research and Practice,* In: E. Burke and H. Rudov (eds.), Proceedings of PATAT '06 (selected papers), Lecture Notes in Computer Science 3867, pages 3-23 (2007).
3. A. Schaerf, *A survey of automated Timetabling,* Artificial Intelligence Review, volume 13(2), pages 87-127 (1999).