AI-Based Tool for Curriculum-Based Course Timetabling at the University of Potsdam

Christian Dohrmann[1,2], Ulrike Lucke[1], Torsten Schaub[1], and Sebastian Schellhorn[1]

[1] University of Potsdam, Germany
{christian.dohrmann, ulrike.lucke, torsten.schaub, sebastian.schellhorn}@uni-potsdam.de
[2] Leibniz Institute for Science and Mathematics Education Kiel

Abstract

Education is a fundamental component of universities and is linked with the non-trivial possibly time-consuming task of creating suitable timetables for teaching. The corresponding problem is known as Curriculum-Based Course Timetabling, which takes into account various constraints while seeking a conflict-free timetable. We addressed this problem by following a knowledge representation and reasoning approach, describing constraints and feasible assignments of courses to time slots. Possible assignments are gathered through a prototypical web interface, and the corresponding constraints are respected by automatically finding a timetable using an AI. We discuss the resolved redundancies, time saved and potential future benefits when comparing the traditional and new approach regarding the needs of teachers, students and administration at the University of Potsdam.

1 Motivation

Education is a fundamental component of universities, usually addressed by offering a variety of courses that may include lectures, exercises, seminars, projects, etc., within defined study programs. In order to obtain optimal timetables, all offered courses must be assigned to a time slot and room, while respecting some hard and soft constraints. A time slot is a combination of a day and a time.

The associated problem is known as Curriculum-Based Course Timetabling (CB-CTT; [4]), which takes into account various constraints when seeking a conflict-free timetable. A solution to a CB-CTT problem is an assignment of courses to time slots and rooms, such that each corresponding hard constraint is satisfied. For instance, there should be no more than one course assigned to a room and time slot, except when explicitly stated. An optimal solution is a solution with minimal penalties regarding soft constraints.

Traditionally, solving CB-CTT problems involve human efforts, consuming a significant amount of time and requiring numerous agreements. Especially when multiple individuals are involved in developing timetables, redundancies and conflicts among the resulting plans may arise. In addition, the obtained timetables are often not guaranteed to be optimal and may possibly violate soft or hard constraints.

In our work, we model the relevant soft and hard constraints within the context of the University of Potsdam (UP) to obtain conflict-free and optimal timetables derived by an AI. Thanks to its simple modeling language and high-performance solving capabilities, Answer Set Programming (ASP; [3]) is a well-suited approach for knowledge representation and reasoning in modeling CB-CTT problems, as demonstrated by [2]. We developed a prototypical web interface allowing planners and lecturers to enter availabilities for teaching times and suitable rooms regarding courses. This interface can automatically generate a timetable within seconds, ensuring compliance with the required constraints among curricula across all involved institutes and planners.

The idea is to find a solution from the time slots specified by lecturers rather than permitting any course to potentially take place at any time slot. As a goal, the obtained timetables should automatically result into corresponding room reservations at universities course catalog.
The remaining article first compares the traditional and new appraoch of solving CB-CTT problems at the UP in Section 2. In Section 3, we introduce basics of our knowledge representation and reasoning approach and illustrate modeled constraints. Afterwards, in Sections 4 and 5, we sketch resolved redundancies, time savings and possible future benefits.

2 Overall Approach

In the following, we describe the traditional and new approach of finding a timetable and link it to the course catalog at UP. To this end, let us first describe the structure and relationships among curricula, modules, examination tasks and courses.
A cohort of students is given by a corresponding program and semester wrt the curriculum. Each curriculum is defined by referring to modules needed to be accomplished. Modules specify the number and type of courses and course components, along with their primary and secondary examination tasks. For simplicity, in the following, we use the term *courses* to refer to both courses and course components, explicitly indicating the distinction when necessary. Courses are linked to, possibly several, secondary and sometimes primary examination tasks within a set of modules. As modules are associated with possibly several curricula, the linkage of examination tasks to courses induces involved curricula and cohorts possibly participate to a course. Whenever a module occurs in several curricula, possibly of different institutes or faculties, then the objective to find a conflict-free timetable becomes more complicated.
In the traditional case, lectures with a large number of participants are planned first, due to a limited number of large lecture halls and imports/exports of courses among curricula. The resulting timetable regarding large lectures needs a lot of communication and agreements among several planners to ensure a conflict-free timetable for involved cohorts and lecturers. Especially, in the case of degree programmes that aim to train teachers with allowed combinations of subjects, UP has developed a so called time frame model to prevent conflicts among courses of most common combinations of topics[1]. On top of scheduled large lectures, the remaining courses are planned by the respective planners of each institute to achieve suitable timetables. A timetable should avoid conflicts regarding multiple use of rooms, lecturers and cohorts as well as respecting particular demands of courses, like technical equipment. A timetable is considered more optimal, if it respects for instance traveling time of consecutive courses, teaching load per day and gaps among courses of a day aiming on same cohort or lecturer, respectively. Once a solution is found, each planner must manually copy it, course by course, to the universities course catalog, link each course to examination tasks of particular modules and book a room for the assigned time slot.
Our approach aims to minimize time-consuming communication among planners and lecturers, as well as reduce redundancies when a solution is found but needs to be copied to the course catalog, linked to modules, and booked into rooms. To this end, we developed a prototypical web interface that enables planners and lecturers to collaboratively create courses and course components, providing all necessary information to facilitate finding a feasible solution. In more detail, for each course, the system requests a name, a list of linked modules, specification of whether it is offered in the winter or summer term, and the designation of a responsible planner. The web interface allows to state the type, e.g. lecture or seminar, the expected number

[1] https://www.uni-potsdam.de/en/studium/studying/organizing-your-studies/potsdams-time-frame-model

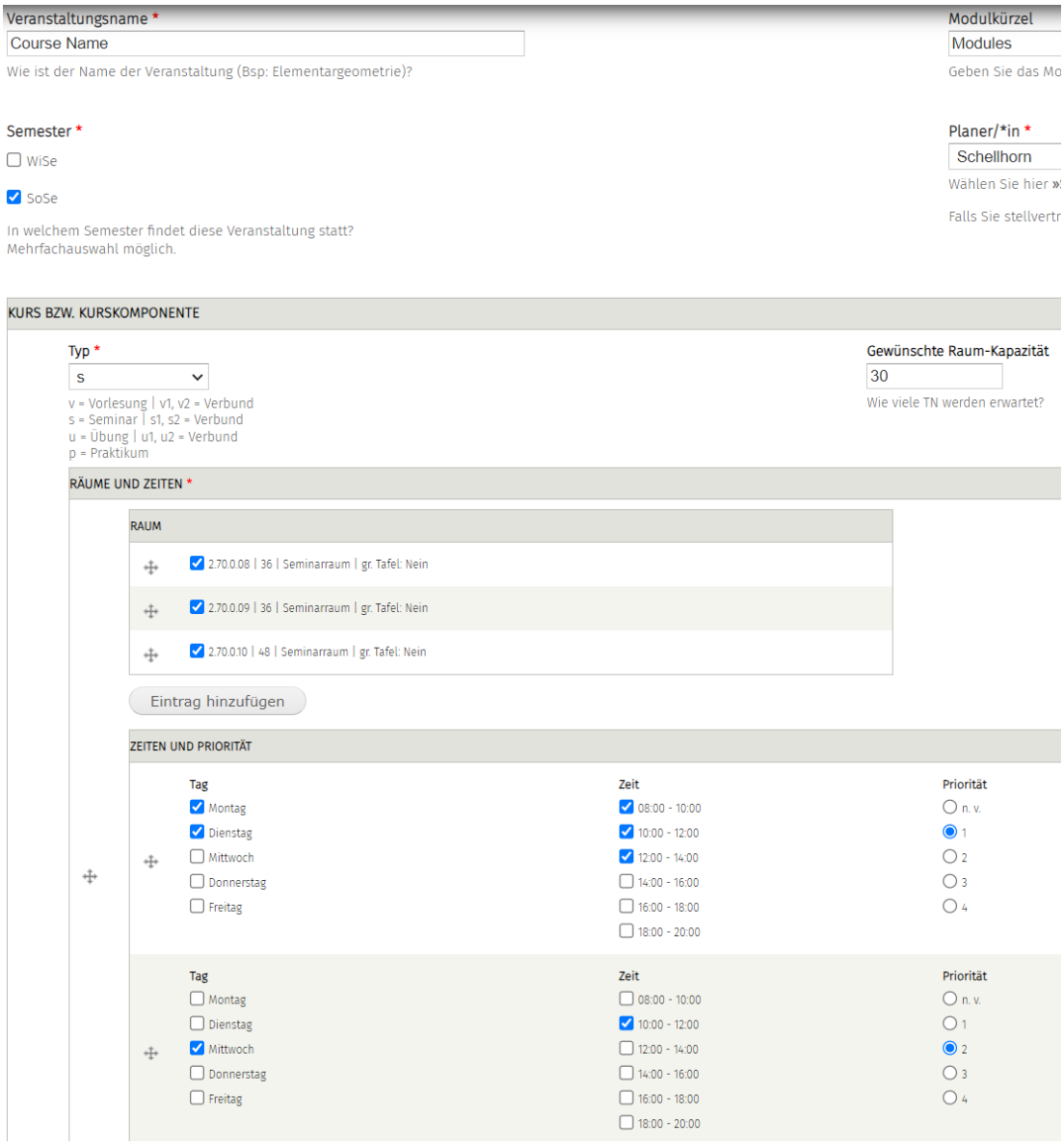

Figure 1: Web interface inducing 18 availabilities of highest priority 1 and three availabilities of priority 2 for a seminar with expected number of 30 participants.

of participants and any specific equipment requirements, e.g. large board, for each course component. Furthermore, for each course component, particular rooms, days and times, along with a priority can be selected. Through the cross product of days, times and rooms possible prioritized availabilities are established for each course component. See figure 1 illustrating parts of the prototypical web interface. We identified and provided four pairwise relations to capture significant constraints among course components:

- Two course components are not allowed to take place in parallel, e.g. obligatory lectures for the same cohort.
- Two course components have to take place in parallel (same time slot, but different rooms), e.g. two groups of exercises for the same course.
- Two course components have to take place simultaneous (same time slot and room), e.g. hosting two seminars, with different names but sharing major content, in the same room.
- Two course components have to take place consecutive, e.g. having an exercise directly after the corresponding lecture.

These four relations allow us to pose certain structures among course components, which enable or disable underlying constraints as elaborated in the next section.
The corresponding planner verifies the consistency and correctness of the input data for the web interface. The input data, along with a logic representation of necessary constraints, is given to an AI problem solver, which then searches for an optimal timetable. Intuitively, the data collected by the web interface can be understood as a particular problem instance and the logic representation of constraints can be understood as a general and persistent problem description. The output is a visualization of the resulting timetable and is double-checked by planners. As an example of a resulting timetable see figure 2.
Note that the new approach aligns with the traditional idea of splitting the process into two steps: initially finding a timetable for large lectures and subsequently completing it to a timetable for each institute. The resulting timetable for each institute must be (automatically) transferred to the university's course catalog, taking into account linked modules, lecturers, time slots, and rooms. The final step of automatically integrating the obtained timetable into the course catalog is not yet implemented in the prototype but will eventually eliminate many redundancies.

3 Technical Solution: ASP Paradigm

ASP [3, 6] is an approach to declarative problem solving and belongs to the area of knowledge representation and reasoning. The roots of ASP go back to logic programming, nonmonotonic reasoning and constraint satisfaction [15]. The idea of ASP is to describe the problem using a formal representation, rather than instructing a computer how to solve the problem. Figure 3 illustrates this approach [13].
An easy and human-readable modeling language [7] is used to create a logic program modeling the original problem. By utilizing a problem solver such as *clingo* [12], a solution, i.e. a stable model [14], of the problem is found and has to be interpreted by the user. Due to its simple but rich high-level modeling language, its elaboration-tolerance, and its high-performance solving capacities, ASP becomes a well suited approach to model real-world problems.
The range of applications for modeling and solving combinatorial search problems with ASP spans across various domains, including planning, scheduling, configuration, probabilistic reasoning, diagnosis and repair, classification, query answering, explanation generation, multi-agent systems, natural language processing, computational biology, music composition, model checking and robotics [8]. In Terms of industrial applications, ASP has been successfully employed for solving (automated product) configuration [18, 16], planning [17, 19], scheduling [1], timetabling [2] and other problems [10]. Especially for industrial applications, tackled by a declarative approach like ASP, significant improvements wrt implementation, maintenance

Figure 2: Part of the resulting timetable containing about 200 course components at the Faculty of Science for summer term 2024. Green indicates lectures, red exercises, blue seminars and yellow projects.

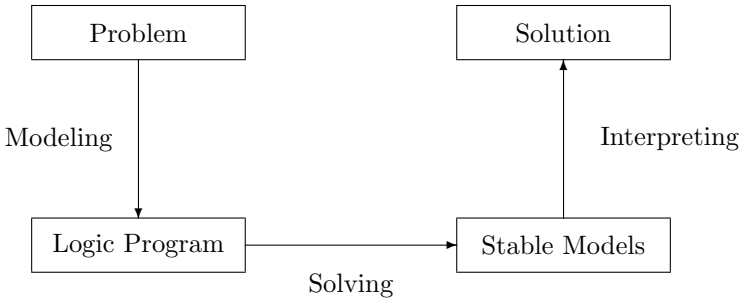

Figure 3: Schematic representation of declarative problem solving process.

costs and human-machine interaction were shown. For instance, Siemens reported saving more than 80% of maintenance costs by using a knowledge representation and reasoning approach, as mentioned in [9].

3.1 Modeling with ASP

ASP provides a powerful high-level modeling language that supports recursive definitions, default negation for dealing with the absence of information, disjunctions, aggregates, weight constraints, optimization statements and external atoms. In the following, we provide a brief introduction to the basics of the modeling language of ASP [7] and the stable models semantics of logic programs [14].

Terms are either constants, variables, arithmetic terms or functional terms. Constants are denoted by strings starting with lowercase letter, quoted strings or integers. Variables are denoted by strings starting with uppercase letter. A predicate atom, is of form $p(t_1, \ldots, t_n)$, where p is a predicate name, $t_1, \ldots, t_n$ are terms and $n \geq 0$ is the arity of the predicate atom. Predicate atoms of arity 0 are represented by its name without parentheses. In the following, predicate atoms are called atoms for short and are denoted by a. An atom is called ground and can be understood as propositional, if all its variables are substituted by constants. A literal is an atom a or its default negation $not\ a$. Default negation, also known as negation as failure, refers to the absence of information, whereas "classical" negation induces the presence of negated information.

A rule r is of form

$$a_0 \leftarrow a_1, \ldots, a_n, not\ a_{n+1}, \ldots, not\ a_m$$

where $0 \leq n \leq m$ and each a_i is an atom for $1 \leq i \leq m$. Let $head(r) = a_0$ and $body(r) = \{a_1, \ldots, a_n, not\ a_{n+1}, \ldots, not\ a_m\}$ the head and the body of a rule r, respectively. For a set of literals X, let $X^+ = \{a \mid a \in X\}$ and $X^- = \{a \mid not\ a \in X\}$. Intuitively, the $head(r)$ of a rule r must hold, if the $body(r)$ holds. The body $body(r)$ of rule r holds, if atoms of $body(r)^+$ are provably true and atoms of $body(r)^-$ are possibly false. A rule r is called fact, whenever $body(r) = \emptyset$.[2] A rule r is called (integrity) constraint, whenever $head(r) \cap body(r)^- \neq \emptyset$. Integrity constraints are used to eliminate unintended solution candidates, whenever its body is satisfied. As an example, consider rule

```
conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))) :-
                          book(42, ("AI", v)), book(43, ("AI", u)),
                          consecutive(("AI", v), ("AI", u)),
                          availability_day(42, fr),
                          not availability_day(43, fr).
```

deriving a conflict among two course components of "AI" lecture and exercise that shall take place consecutive, whenever they are scheduled on a different day than friday.[3]

A logic program is a finite set of rules. A set of ground atoms X is a model of a logic program P, if for every $r \in P$ holds $head(r) \in X$ whenever $body(r)^+ \subseteq X$ and $body(r)^- \cap X = \emptyset$. The stable model of a program P is defined relative to a set of atoms X, by the so called reduct P^X, defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P, body(r)^- \cap X = \emptyset\}.$$

[2] Usually, we drop $\leftarrow$ in this case.

[3] In the syntax of an ASP encoding '$\leftarrow$' is represented by ': −' and each rule terminates by a period '.'.

X is a stable model of a program P, if X is the $\subseteq$-minimal model of P^X.
As an example consider the following program P_1

```
1  room("2.70.0.10").
2  room("2.70.0.11").
3  room_capacity("2.70.0.10", 48).
4  room_capacity("2.70.0.11", 60).
5  course(("AI", v)).
6  course(("AI", u)).
7  course_size(("AI", v), 50).
8  course_size(("AI", u), 50).
9  consecutive(("AI", v), ("AI", u)).
10 availability_course(42, ("AI", v)).
11 availability_room(42, "2.70.0.11").
12 availability_day(42, fr).
13 availability_time(42, 10).
14 availability_course(43, ("AI", u)).
15 availability_room(43, "2.70.0.10").
16 availability_day(43, mo).
17 availability_time(43, 12).
18 availability_course(44, ("AI", u)).
19 availability_room(44, "2.70.0.11").
20 availability_day(44, fr).
21 availability_time(44, 12).
22 book(42, ("AI", v)) :- availability_course(42, ("AI", v)).
23 book(43, ("AI", u)) :- availability_course(43, ("AI", u)),
24                         not book(44, ("AI", u)).
25 book(44, ("AI", u)) :- availability_course(44, ("AI", u)),
26                         not book(43, ("AI", u)).
27 conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))) :-
28                              book(42, ("AI", v)), book(43, ("AI", u)),
29                              consecutive(("AI", v), ("AI", u)),
30                              availability_day(42, fr),
31                              not availability_day(43, fr).
32 conflict :- conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))),
33              not conflict.
```

where lines 1-21 are facts setting rooms, room capacities, course components with expected number of participants, stating "AI" lecture and exercise as consecutive as well as declaring possible time slots and rooms for each course component. Line 22 sets "AI" lecture to time slot with identifier 42, since there is no other option. Lines 23-26 state that either time slot with identifier 43 or 44 has to be taken for the "AI" exercise. Lines 27-31 derive a conflict predicate, whenever "AI" lecture and exercise are not at same day and thus not consecutive. Lines 32-33 form an integrity constraint ruling out models containing conflict predicate derived by lines 27-31. Since atoms of lines 1-21 are facts, they belong to any model of P_1. Let X be the set containing atoms of lines 1-21. Note that X is not a model of P_1, since the body of rule in line 22 is satisfied, but not its head. Sets

$X \cup \{$`book(42, ("AI", v)), book(43, ("AI", u)),`
`conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u)))), conflict`$\}$

and

$X \cup \{$`book(42, ("AI", v)), book(44, ("AI", u))`$\}$

are models of P_1, but the latter one is the only stable model of P_1, since first model is not $\subseteq$-minimal under reduct wrt P_1, due to

$$X \cup \{\texttt{book(42, ("AI", v)), book(43, ("AI", u)),}\\ \texttt{conflict(consecutive, ((42, ("AI", v)), (43, ("AI", u))))}\}.$$

The basic approach of modeling a problem by an ASP encoding follows a generate-and-test methodology, also known as guess-and-check, inspired by intuitions on *NP* problems. NP problems are problems, where no algorithm is known to solve them in polynomial time. Intuitively, NP problems are hard to solve, since the needed time may grow exponentially. In the case of CB-CTT for a generate-and-test methodology, one could think of generating any timetable as a model and eliminating certain models by imposing specific (hard) constraints.

3.2 Modeling Curriculum-Based Course Timetabling

Since a course may consists of several course components of possibly different types, e.g. a lecture and an exercise, we identify a course component by a unique tuple of a name and its type, e.g. lecture *("AI", v)* and exercise *("AI", u)* of the course named Artificial Intelligence. We modeled the following hard (H0-8) and soft (S0-13) constraints.

H0. Components: No component of a course should take place in parallel to its corresponding lecture, except if explicitly stated.

H1. Courses: One availability for each course component must be assigned to a time slot and room.

H2. Cohort: Obligatory lectures of the same cohort must be all scheduled in different time slots, except explicitly stated to take place in parallel.

H3. RoomOccupancy: Two course components cannot take place in the same room and time slot, except explicitly stated.

H4. Lecturer: Course components sharing a lecturer cannot be scheduled in parallel, except explicitly stated.

H5. NotParallel: Avoiding two course components to take place in parallel, whenever explicitly stated.

H6. Parallel: Two course components have to take place in parallel, whenever explicitly stated.

H7. Simultaneous: Two course components have to take place simultaneously, whenever explicitly stated.

H8. Consecutive: Two course components have to be scheduled consecutively, whenever explicitly stated.

S0. Availability: For each course component, try to serve prioritized availability on time slot and room. The penalty points reflect corresponding priorities.

S1. RoomCapacity: For each course component, penalty points for the number of students that are expected to attend the course minus the number of seats in the corresponding room are imposed on each violation.

S2. Cohort: Course components addressed to the same cohort should be scheduled in different time slots, except explicitly stated to be in parallel. Each violation counts as a penalty point.

S3. Gaps: For a cohort as well as a lecturer, corresponding course components should be scheduled in time slots as close as possible. The penalty regarding two courses sharing a cohort or lecturer and a day is given by subtracting the earlier time from the later time.

S4. RoomStability: Two course components stated to be consecutive should be booked in the same room. The penalty points reflect each violation.

S5. MaxLoad: For a cohort as well as lecturer the number of corresponding course components per day should be lower or equal to a given maximum. The penalty points reflect the number of courses beyond the maximum.

S6. TravelTime: For a cohort as well as a lecturer, traveling time between rooms with two adjacent course components should be as small as possible. The penalty is reflected by the traveling time itself.

S7-13. RoomSuitability: Some course components prefer particular equipment like a large board, projector, computer, microphone, camera, media table or whiteboard. Each violation counts as a penalty point.

Note that some constraints are similar to those presented in [2], others are modified or new. Due to lack of space, we omit the corresponding logic program here.[4] Optimization aims to minimize penalty points associated with the aforementioned soft constraints.
The system of [2] reads instances in a standard input format [5], translates them into ASP facts and assigns potentially any course to any time slot. In contrast, we employed a direct modeling approach tailored to the constraints and requirements of the UP, which partially is not covered by the standard input format of the CB-CTT community. As a design decision, we reduced the search space to collected availabilities only, rather than of checking for all possible time slots and rooms for each course component.

4 Results and Impact

In [2], it was demonstrated that addressing CB-CTT problems through an ASP-based approach outperforms other state-of-the-art approaches and thus makes it well-suited to tackle CB-CTT problems. Our real world approach is still at a prototypical stage and it is lacking (automated) access to some data needed to apply all presented and modeled constraints.
For the winter semester of 2023/24, an optimal and conflict-free plan was found for approximately 160 courses aimed at large lecture halls of the Faculty of Science, additional 90 courses of the Institute of Mathematics and 100 courses of the Institute of Computer Science, incorporating constraints H1, H3, H5, S0, S1 and S7 from the above ASP-based modeling. Analogously, for the summer semester of 2024, an optimal and conflict-free plan was found involving about 200 courses and constraints H0, H1, H3, H5, H6, H7, H8, S0, S1, S4 and S7. The *clingo* solver was integrated into the web interface and required approximately 170 seconds for the winter and less than one second for the summer semester, respectively, to schedule the curricula for the faculty. Finding plans for each institute took less than a second. The time required to solve these problems relative to the number of involved courses highlights the exponential complexity inherent in CB-CTT problems. In practice, solving a problem with a larger number of courses may require days until finding an optimal solution and thus become possibly inconvenient in usage. For this reason, it is crucial to maintain a workflow that breaks down the planning task into several smaller tasks, similar to the traditional approach.
A major benefit of the presented AI-based approach is its fairness, transparency and elaboration tolerance. Thanks to its elaboration tolerance, the existing encoding can be reused for finding a plan for each upcoming semester as it only depends on given courses. It is possible to add new constraints or relax existing ones whenever needed to address the particular demands of institutes or faculties. Planners and lecturers can set a prioritized selection of availabilities

[4] The particular ASP encoding representing above constraints can be found at GitHub: https://github.com/schellhorn/CBCTT-UP.

and preferences for upcoming teaching periods, resulting in an optimal and provably conflict-free timetable. Obtaining a conflict-free timetable for a faculty in a matter of seconds, while respecting dependencies among the several institutes at the same time, resolves a significant amount of communication overhead and conflicts inherent in the traditional approach. Thus, our approach provides planners with the opportunity to allocate their additional available time to other tasks.
Once the integration of the resulting timetable into the university's course catalog is done automatically, even more redundancies are resolved, and a considerable amount of additional human capacities become available. Due to our current prototypical state, we have not yet conducted a performance comparison with other approaches[11].

5 Future Work

In future, we plan to implement an interface that allows us to access data such as previous course catalogues, curricula, and their modules. Furthermore, we plan to develop an interface that allows us to automatically push final timetables to the university's course catalog.
Due to the reduced search space by concerning availabilities only, we may run into conflicts and cannot find any plan. To address the non-trivial task to resolve conflicts, we plan to provide opportunities to the user to analyze and resolve conflicts whenever needed. Another feature one could think about to take advantage of older plans, is to allow an option to copy previous availabilities adapted to the result of a previous solution. In a next step, we plan to expand the number of users and involve them in a feedback loop to enhance the usage of the system and identify additional needs. In the long run, one could think about adding functionalities such as drag and drop to manually incorporate smaller changes to a calculated timetable, while preserving consistency regarding hard and soft constraints.
When establishing the mentioned workflow to obtain conflict-free timetables, one could measure the average amount of time needed per planner using the traditional and new approach, respectively. Moreover, one could compare the quality of resulting timetables regarding the desired hard and soft constraints. Comparing the quality and time efficiency of both approaches may highlight the usage and impact of our new approach using AI of knowledge representation and reasoning.
The data about availabilities of different types of course components in various time slots provides insights into the needs of room infrastructure. One could consider possibly leveraging entered availabilities and the number of participants to derive the location, size, and quantity of needed lecture halls and seminar rooms when planning to build new ones.
Finally, our approach has significant potential to eliminate redundant workflows, promotes trustworthy collaborations among institutes, save money, and elevate administrative planning tasks to the next level.

References

[1] M. Alviano, C. Dodaro, and M. Maratea. An advanced answer set programming encoding for nurse scheduling. In F. Esposito, R. Basili, S. Ferilli, and F. Lisi, editors, *Proceedings of the Sixteenth International Conference of the Italian Association for Artificial Intelligence*, volume 10640, pages 468–482, 2017.

[2] M. Banbara, K. Inoue, B. Kaufmann, T. Okimoto, T. Schaub, T. Soh, N. Tamura, and P. Wanko. teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, 275(1):3–37, 2019.

[3] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[4] A. Bettinelli, V. Cacchiani, R. Roberti, and Paolo Toth. An overview of curriculum-based course timetabling. *TOP*, 23(2):313–349, 2015.

[5] A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf. Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, 194(1):59–70, 2012.

[6] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[7] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, and T. Schaub. ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309, 2020.

[8] E. Erdem, M. Gelfond, and N. Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

[9] A. Falkner, G. Friedrich, A. Haselböck, G. Schenner, and H. Schreiner. Twenty-five years of successful application of constraint technologies at siemens. 37(4):67–80, 2016.

[10] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. Teppan. Industrial applications of answer set programming. *Künstliche Intelligenz*, 32(2-3):165–176, 2018.

[11] T. Feutrier. *Landscape Analysis and Solver Reconfiguration for the Curriculum-Based Course Timetabling*. Theses, Université de Lille, December 2023.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko. Theory solving made easy with clingo 5. In M. Carro and A. King, editors, *Technical Communications of the Thirty-second International Conference on Logic Programming (ICLP'16)*, volume 52 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[13] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[15] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[16] K. Herud, J. Baumeister, O. Sabuncu, and T. Schaub. Conflict handling in product configuration using answer set programming. In *Proceedings of the Fifteenth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'22)*, volume 3193 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[17] Y. Jiang, S. Zhang, P. Khandelwal, and P. Stone. Task planning in robotics: an empirical comparison of PDDL- and ASP-based systems. *Frontiers of Information Technology and Electronic Engineering*, 20(3):363–373, 2019.

[18] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In G. Gupta, editor, *Practical Aspects of Declarative Languages, First International Workshop, PADL '99, San Antonio, Texas, USA, January 18-19, 1999, Proceedings*, volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 1999.

[19] T. Son, E. Pontelli, M. Balduccini, and T. Schaub. Answer set planning: A survey. *Theory and Practice of Logic Programming*, 23(1):226–298, 2023.