# *teaspoon*: Solving the Curriculum-Based Course Timetabling Problems with Answer Set Programming

**Mutsunori Banbara · Katsumi Inoue · Benjamin Kaufmann · Torsten Schaub · Takehide Soh · Naoyuki Tamura · Philipp Wanko**

**Abstract** Answer Set Programming (ASP) is an approach to declarative problem solving, combining a rich yet simple modeling language with high performance solving capacities. We here develop an ASP-based approach to *Curriculum-Based Course Timetabling* (CB-CTT), one of the most widely studied course timetabling problems. The resulting *teaspoon* system reads a CB-CTT instance of a standard input format and converts it into a set of ASP facts. In turn, these facts are combined with a first-order encoding for CB-CTT solving, which can subsequently be solved by any off-the-shelf ASP systems. We establish the competitiveness of our approach by empirically contrasting it to the best known bounds obtained so far via dedicated implementations.

**Keywords** Educational Timetabling · Course Timetabling · Answer Set Programming

## 1 Introduction

*Educational timetabling* [14, 28, 36] is generally defined as the task of assigning a number of events, such as lectures and examinations, to a limited set of timeslots (and perhaps rooms), subject to a given set of hard and soft constraints. Hard constraints must be strictly satisfied. Soft constraints must not necessarily be satisfied but the overall number of violations should

The work was partially funded by JSPS (KAKENHI 15K00099), JSPS (KAKENHI 16H02803), and DFG grants SCHA 550/9-2 and SCHA 550/11-1.

M. Banbara · T. Soh · N. Tamura
Kobe University, 1-1 Rokko-dai Nada-ku Kobe Hyogo, 657-8501, Japan
E-mail: {banbara@, soh@lion., tamura@}kobe-u.ac.jp

K. Inoue
National Institute of Informatics, 2-1-2 Hitotsubashi Chiyoda-ku Tokyo, 101-8430, Japan
E-mail: inoue@nii.ac.jp

B. Kaufmann · P. Wanko
Universität Potsdam, August-Bebel-Strasse 89, D-14482 Potsdam, Germany
E-mail: {kaufmann, wanko}@cs.uni-potsdam.de

T. Schaub
Universität Potsdam, August-Bebel-Strasse 89, D-14482 Potsdam, Germany
Inria – Centre de Rennes Bretagne Atlantique, France
E-mail: torsten@cs.uni-potsdam.de

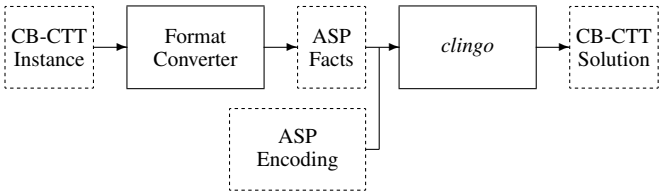


**Fig. 1** Architecture of *teaspoon*.

be minimal. The educational timetabling problems can be classified into three categories: *school timetabling*, *examination timetabling*, and *course timetabling*. In this paper, we focus on *curriculum-based course timetabling* (CB-CTT) [8], one of the most studied course timetabling problems, as well as *post-enrollment course timetabling*.

The CB-CTT problems have been used in the third track of the second international timetabling competition (ITC-2007) [15,32]. A web portal[1] for CB-CTT has been actively maintained by the ITC-2007 organizers [10]. The web site has provided necessary infrastructures for benchmarking such as validators, data formats, problem instances, solutions in different formulations (uploaded by researchers), and visualizers. All problem instances on the web are based on real data from various universities. The best known bounds on the web have been obtained by various methods including the winner algorithm of ITC-2007: metaheuristics-based algorithms [1,16,17,24,29], Integer Programming [27], hybrid methods [33], SAT/MaxSAT [2], and many others.

However, each method has strength and weakness. Metaheuristics-based dedicated implementations can quickly find better upper bounds, but cannot guarantee their optimality. Although complete methods such as SAT can guarantee the optimality, it is costly to implement a dedicated encoder from the CB-CTT problems in SAT. Integer Programming has been widely used for CB-CTT solving, but in general it does not scale to large instances in complex formulations. It is therefore particularly challenging to develop a universal timetabling solver which can efficiently find optimal solutions as well as better bounds for a wide range of CB-CTT instances in different formulations at present.

Answer Set Programming (ASP; [7,25,35]) is an approach to declarative problem solving. Recent advances in ASP open up a successful direction to extend logic programming to be both more expressive as well as more effective. ASP provides a rich language and is well suited for modeling combinatorial (optimization) problems in Artificial Intelligence and Computer Science. Recent remarkable improvements in the effectiveness of ASP systems have encouraged researchers to use ASP for solving problems in diverse areas, such as automated planning, constraint satisfaction, model checking, music composition, robotics, system biology, etc. However, so far, little attention has been paid to using ASP for timetabling.

In this paper, we describe an ASP-based approach for solving the CB-CTT problems and present the resulting *teaspoon* system. The *teaspoon* system reads a CB-CTT instance of a standard input format [10] and converts it into ASP facts. In turn, these facts are combined with a first-order encoding for CB-CTT solving, which is subsequently solved by an off-the-shelf ASP system, in our case *clingo* [2]. Figure 1 shows the *teaspoon* architecture.

The high-level approach of ASP has obvious advantages. First, the problems are solved by general-purpose ASP systems rather than dedicated implementation. Second, the elabo-

[1] `http://tabu.diegm.uniud.it/ctt/`

[2] ASP system *clingo* is a monolithic combination of the grounder *gringo* with the solver *clasp*.

ration tolerance of ASP allows for easy maintenance and modifications of encodings. And finally, it is easy to experiment with advanced techniques in ASP solving such as core-guided optimization, domain heuristics, and portfolios of prefabricated expert configurations [20]. However, the question is whether the high-level approach of *teaspoon* matches the performance of dedicated systems. We empirically address this question by contrasting the performance of *teaspoon* with the best known bounds on the CB-CTT web portal.

From an ASP perspective, we showed in previous work [6] that ASP's modeling language is well-suited for course timetabling by providing a compact encoding for CB-CTT. However, at the same time, we observed that a simple branch-and-bound optimization strategy is insufficient to decrease the upper bounds of large instances in complex formulations. In this paper, we provide insights into how more advanced solving techniques can be used to overcome this practical issue. The ASP encoding implementing CB-CTT solving with *teaspoon* is an extension of our previous encoding with the following features: (a) a collection of optimized encodings for soft constraints, (b) easy composition of different formulations, (c) support for multi-criteria optimization based on lexicographic ordering, (d) reusing legacy timetables with multi-shot ASP solving.

Our encoding is given in the *gringo* 4 language [19]. Although we provide a brief introduction to ASP and its basic language constructs in Section 3, we refer the reader to the literature [7, 21] for a comprehensive treatment of ASP.

## 2 Curriculum-based Course Timetabling

The basic entities of the CB-CTT problem are *courses*, *rooms*, *days*, and *periods* per day. A *timeslot* is a pair composed of a day and a period. A *curriculum* is a group of courses that shares common students. The CB-CTT problem is defined as the task of assigning all lectures of each course into a weekly timetable, subject to a given set of constraints: *hard* constraints ($H_1$–$H_4$, see below) and *soft* constraints ($S_1$–$S_9$). The former must be strictly satisfied. The latter are not necessarily satisfied but the sum of their violations should be minimal. From the viewpoint of violations, the soft constraints can be divided into two types: the soft constraints with *constant cost* ($S_3$ and $S_7$–$S_9$) and the soft ones with *calculated cost* ($S_1$–$S_2$ and $S_4$–$S_6$). The difference is that for those with constant cost only one penalty point is imposed on each violation, whereas many penalty points calculated dynamically in accordance with each violation are imposed for those with calculated cost. A *feasible solution* of the problem is an assignment in which all lectures are assigned to a timeslot and a room, so that the hard constraints are satisfied. The objective of the problem is to find a feasible solution of minimal penalty costs. The following definitions are based on [10].

- $\boldsymbol{H_1}$. **Lectures**: All lectures of each course must be scheduled, and they must be assigned to distinct timeslots.
- $\boldsymbol{H_2}$. **Conflicts**: Lectures of courses in the same curriculum or taught by the same teacher must be all scheduled in different timeslots.
- $\boldsymbol{H_3}$. **RoomOccupancy**: Two lectures cannot take place in the same room in the same timeslot.
- $\boldsymbol{H_4}$. **Availability**: If the teacher of the course is unavailable to teach that course at a given timeslot, then no lecture of the course can be scheduled at that timeslot.
- $\boldsymbol{S_1}$. **RoomCapacity**: For each lecture, the number of students that attend the course must be less than or equal the number of seats of all the rooms that host its lectures. The penalty points, reflecting the number of students above the capacity, are imposed on each violation.

**Table 1** Problem Formulations

| Constraint | UD1 | UD2 | UD3 | UD4 | UD5 |
|---|---|---|---|---|---|
| $H_1$. Lectures | H | H | H | H | H |
| $H_2$. Conflicts | H | H | H | H | H |
| $H_3$. RoomOccupancy | H | H | H | H | H |
| $H_4$. Availability | H | H | H | H | H |
| $S_1$. RoomCapacity | 1 | 1 | 1 | 1 | 1 |
| $S_2$. MinWorkingDays | 5 | 5 | - | 1 | 5 |
| $S_3$. IsolatedLectures | 1 | 2 | - | - | 1 |
| $S_4$. Windows | - | - | 4 | 1 | 2 |
| $S_5$. RoomStability | - | 1 | - | - | - |
| $S_6$. StudentMinMaxLoad | - | - | 2 | 1 | 2 |
| $S_7$. TravelDistance | - | - | - | - | 2 |
| $S_8$. RoomSuitability | - | - | 3 | H | - |
| $S_9$. DoubleLectures | - | - | - | 1 | - |

- $S_2$. **MinWorkingDays**: The lectures of each course must be spread into a given minimum number of days. The penalty points, reflecting the number of days below the minimum, are imposed on each violation.
- $S_3$. **IsolatedLectures**: Lectures belonging to a curriculum should be adjacent to each other in consecutive timeslots. For a given curriculum we account for a violation every time there is one lecture not adjacent to any other lecture within the same day. Each isolated lecture in a curriculum counts as 1 violation.
- $S_4$. **Windows**: Lectures belonging to a curriculum should not have time windows (periods without teaching) between them. For a given curriculum we account for a violation every time there is one window between two lectures within the same day. The penalty points, reflecting the length in periods of time window, are imposed on each violation.
- $S_5$. **RoomStability**: All lectures of a course should be given in the same room. The penalty points, reflecting the number of distinct rooms but the first, are imposed on each violation.
- $S_6$. **StudentMinMaxLoad**: For each curriculum the number of daily lectures should be within a given range. The penalty points, reflecting the number of lectures below the minimum or above the maximum, are imposed on each violation.
- $S_7$. **TravelDistance**: Students should have the time to move from one building to another one between two lectures. For a given curriculum we account for a violation every time there is an *instantaneous move*: two lectures in rooms located in different building in two adjacent periods within the same day. Each instantaneous move in a curriculum counts as 1 violation.
- $S_8$. **RoomSuitability**: Some rooms may be not suitable for a given course because of the absence of necessary equipment. Each lecture of a course in an unsuitable room counts as 1 violation.
- $S_9$. **DoubleLectures**: Some courses require that lectures in the same day are grouped together (*double lectures*). For a course that requires grouped lectures, every time there is more than one lecture in one day, a lecture non-grouped to another is not allowed. Two lectures are grouped if they are adjacent and in the same room. Each non-grouped lecture counts as 1 violation.

The *formulation* is defined as a specific set of soft constraints together with the weights associated with each of them. In ITC-2007, the CB-CTT problem is formulated as a combinatorial optimization problem whose objective function is to minimize the weighted sum of penalty points. Until now five formulations have been proposed: UD1–UD5. UD1 is a basic

formulation [16]. UD2 is a formulation used in ITC-2007 [15]. To capture more different scenarios, UD3, UD4, and UD5 are proposed recently [10]. These new formulations focus on student load (UD3), double lectures (UD4), and travel cost (UD5), respectively. Table 1 shows the weights associated with each soft constraint for all formulations. The symbol 'H' indicates that the constraint is a hard constraint. The symbol '-' indicates that the constraint is not included in the formulation.

## 3 Answer Set Programming

Answer Set Programming (ASP; [7,25,35]) is a popular tool for declarative problem solving due to its attractive combination of a high-level modeling language with high-performance search engines. In ASP, problems are described as logic programs, which are sets of rules of the form

$$\texttt{a}_0 \texttt{ :- a}_1\texttt{,...,a}_m\texttt{,not a}_{m+1}\texttt{,...,not a}_n$$

where each $\texttt{a}_i$ is a propositional atom and `not` stands for *default negation*. We call a rule a *fact* if $n = 0$, and an *integrity constraint* if we omit $\texttt{a}_0$. Semantically, a logic program induces a collection of so-called *answer sets*, which are distinguished models of the program determined by answer sets semantics; see [25] for details.

To facilitate the use of ASP in practice, several extensions have been developed. First of all, rules with first-order variables are viewed as shorthand for the set of their ground instances. Further language constructs include *conditional literals* and *cardinality constraints* [35]. The former are of the form $\texttt{a:b}_1\texttt{,...,b}_m$, the latter can be written as $\texttt{s\{c}_1\texttt{,...,c}_n\texttt{\}t}$, where `a` and $\texttt{b}_i$ are possibly default-negated literals and each $\texttt{c}_j$ is a conditional literal; `s` and `t` provide lower and upper bounds on the number of satisfied literals in the cardinality constraint. The practical value of both constructs becomes apparent when used with variables. For instance, a conditional literal like `a(X):b(X)` in a rule's antecedent expands to the conjunction of all instances of `a(X)` for which the corresponding instance of `b(X)` holds. Similarly, `2 {a(X):b(X)} 4` is true whenever at least two and at most four instances of `a(X)` (subject to `b(X)`) are true. A useful[3] shortcut are expressions of the form $\texttt{N = \{c}_1\texttt{,...,c}_n\texttt{\}}$ that binds `N` to the number of satisfied conditional literals $\texttt{c}_j$. Finally, objective functions minimizing the sum of weights $\texttt{w}_j$ of conditional literals $\texttt{c}_j$ are expressed as $\texttt{\#minimize\{w}_1\texttt{:c}_1\texttt{,...,w}_n\texttt{:c}_n\texttt{\}}$. [4]

## 4 The *teaspoon* Approach

We begin with describing *teaspoon*'s fact format of CB-CTT instances and then present an ASP encoding for solving the CB-CTT problems.

**Fact Format.** Listing 1 shows a tiny instance `toy.ectt` written in the '`.ectt`' format, a standard input format of the CB-CTT instances [10]. ASP facts representing `toy.ectt` are shown in Listing 2. The first nine facts in Line 1–2 express the scalar values of each entity. This instance named `Toy` consists of 4 courses, 3 rooms, 2 curricula, 8 unavailability constraints, and 3 room constraints. The weekly timetable consists of 5 days and

[3] Care must be taken whenever such expressions are evaluated during solving (rather than grounding).

[4] Syntactically, each $\texttt{w}_\texttt{j}$ can be an arbitrary term. In fact, often tuples are used rather than singular weights to ensure a multi-set property; in such a case the summation only applies to the first element of selected tuples.

```
Name: Toy
Courses: 4
Rooms: 3
Days: 5
Periods_per_day: 4
Curricula: 2
Min_Max_Daily_Lectures: 2 3
UnavailabilityConstraints: 8
RoomConstraints: 3

COURSES:
SceCosC Ocra 3 3 30 1
ArcTec Indaco 3 2 42 0
TecCos Rosa 5 4 40 1
Geotec Scarlatti 5 4 18 1

ROOMS:
rA 32 1
rB 50 0
rC 40 0

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec

UNAVAILABILITY_CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3
ArcTec 4 0
ArcTec 4 1
ArcTec 4 2
ArcTec 4 3

ROOM_CONSTRAINTS:
SceCosC rA
Geotec rB
TecCos rC

END.
```

**Listing 1** `toy.ectt`: a toy instance

```
1  name("Toy").  courses(4).  rooms(3).  days(5).  periods_per_day(4).  curricula(2).
2  min_max_daily_lectures(2,3).  unavailabilityconstraints(8).  roomconstraints(3).

4  course("SceCosC","Ocra",3,3,30,1).  course("ArcTec","Indaco",3,2,42,0).
5  course("TecCos","Rosa",5,4,40,1).   course("Geotec","Scarlatti",5,4,18,1).

7  room(rA,32,1).  room(rB,50,0).  room(rC,40,0).

9  curricula("Cur1","SceCosC").  curricula("Cur1","ArcTec").  curricula("Cur1","TecCos").
10 curricula("Cur2","TecCos").   curricula("Cur2","Geotec").

12 unavailability_constraint("TecCos",2,0).  unavailability_constraint("TecCos",2,1).
13 unavailability_constraint("TecCos",3,2).  unavailability_constraint("TecCos",3,3).
14 unavailability_constraint("ArcTec",4,0).  unavailability_constraint("ArcTec",4,1).
15 unavailability_constraint("ArcTec",4,2).  unavailability_constraint("ArcTec",4,3).

17 room_constraint("SceCosC",rA).  room_constraint("Geotec",rB).  room_constraint("TecCos",rC).
```

**Listing 2** `toy.lp`: ASP facts representing `toy.ectt`

```
1  assigned("SceCosC",rB,3,0).  assigned("SceCosC",rB,2,2).  assigned("SceCosC",rB,4,2).
2  assigned("ArcTec", rB,3,1).  assigned("ArcTec", rB,0,2).  assigned("ArcTec", rB,1,2).
3  assigned("TecCos", rB,0,1).  assigned("TecCos", rB,0,3).  assigned("TecCos", rB,1,3).
4  assigned("TecCos", rB,2,3).  assigned("TecCos", rB,4,3).  assigned("Geotec", rA,4,1).
5  assigned("Geotec", rA,0,2).  assigned("Geotec", rA,1,2).  assigned("Geotec", rA,2,2).
6  assigned("Geotec", rA,4,2).
```

**Listing 3** Solution (partial answer set) of `toy.lp` in UD2

4 periods per day, where they start from 0. The fact `course(`$C$`,`$T$`,`$N$`,`$MWD$`,`$M$`,`$DL$`)` in Line 4–5 expresses that a course $C$ taught by a teacher $T$ has $N$ lectures, which must be spread into $MWD$ days. The number of students that attend the course $C$ is $M$. The course $C$ requires double lectures if $DL = 1$. The fact `room(`$R$`,`$CAP$`,`$BLD$`)` in Line 7 expresses that a room $R$ located in a building $BLD$ has a seating capacity of $CAP$. The fact `curricula(`$CUR$`, `$C$`)` in Line 9–10 expresses that a curriculum $CUR$ includes a course

```
1  c(C)      :- course(C,_,_,_,_,_).  t(T)        :- course(_,T,_,_,_,_).
2  r(R)      :- room(R,_,_).          cu(Cu)      :- curricula(Cu,_).
3  d(0..D-1) :- days(D).              ppd(0..P-1) :- periods_per_day(P).

5  % H1.Lectures
6  N { assigned(C,D,P) : d(D), ppd(P) } N :- course(C,_,N,_,_,_).

8  % H2.Conflicts
9  :- not { assigned(C,D,P) : course(C,T,_,_,_,_) } 1, t(T), d(D), ppd(P).
10 :- not { assigned(C,D,P) : curricula(Cu,C) } 1, cu(Cu), d(D), ppd(P).

12 % H3.RoomOccupancy
13 1 { assigned(C,R,D,P) : r(R) } 1 :- assigned(C,D,P).
14 :- not { assigned(C,R,D,P) : c(C) } 1, r(R), d(D), ppd(P).

16 % H4.Availability
17 :- assigned(C,D,P), unavailability_constraint(C,D,P).

19 % Additional constraints (can be omitted)
20 :- not { assigned(C,D,P) : c(C) } N, d(D), ppd(P), rooms(N).
```

**Listing 4** Encoding of hard constraints

*C*. The fact `unavailability_constraint(`$C$`,`$D$`,`$P$`)` in Line 12–15, which is used to specify $H_4$, expresses that a course $C$ is not available at a period $P$ on a day $D$. The fact `room_constraint(`$C$`,`$R$`)` in Line 17, which is used to specify $S_8$, expresses that a room $R$ is not suitable for a course $C$.

As an output example, Listing 3 shows an optimal solution with zero cost of the tiny instance `toy.lp` in the UD2 formulation. Each predicate `assigned(`$C$`,`$R$`,`$D$`,`$P$`)` is intended to express that a lecture of a course $C$ is assigned to a room $R$ at a period $P$ on a day $D$. All three lectures of the course `SceCosC` are assigned to the room `rB` at the first period (`0`) on Thursday (`3`), the third period (`2`) on Wednesday (`2`), and the third period (`2`) on Friday (`4`) as can be seen in Line 1.

**First-Order Encoding.** The *teaspoon* encoding of hard constraints ($H_1$–$H_4$) is shown in Listing 4. Each hard constraint can be individually expressed in either one or two rules by using integrity constraints and cardinality constraints. As mentioned above, the predicate `assigned(`$C$`,`$R$`,`$D$`,`$P$`)` expresses that a lecture of a course $C$ is assigned to a room $R$ at a period $P$ on a day $D$, and a solution is composed of a set of these assignments. The predicate `assigned(`$C$`,`$D$`,`$P$`)` dropping $R$ from `assigned(`$C$`,`$R$`,`$D$`,`$P$`)` is also introduced, since we do not always have to take the room information into account to specify the hard constraints except $H_3$.

Given a instance of fact format, the first four rules in Line 1–2 generate `c(C)`, `t(T)`, `r(R)`, and `cu(Cu)` for each course `C`, teacher `T`, room `R`, and curriculum `Cu`. The next two rules in Line 3 generate `d(0)` ... `d(D-1)` and `ppd(0)` ... `ppd(P-1)` express that the days range from `0` to `D-1`, and the periods per day range from `0` to `P-1`.

For $H_1$, the rule in Line 6, for every course `C` having `N` lectures, generates a candidate of assignments at first and then constrains that there are exactly `N` lectures such that `assigned(C,D,P)` holds. For $H_2$, the rule in Line 9 constrains that, for every teacher `T`, day `D`, and period `P`, there is at most one course `C` taught by `T` such that `assigned(C,D,P)` holds. The rule in Line 10 constrains that, for every curriculum `Cu`, day `D`, and period `P`, there is at most one course `C` that belongs to `Cu` such that `assigned(C,D,P)` holds. For $H_3$, the rule in Line 13 generates a solution candidate and then constrains that there is exactly one room `R` such that `assigned(C,R,D,P)` holds if `assigned(C,D,P)` holds. The

```
1  % S1.RoomCapacity
2  penalty("RoomCapacity",assigned(C,R,D,P),(N-Cap)*weight_of_s1) :-
3          assigned(C,R,D,P), course(C,_,_,_,N,_), room(R,Cap,_), N > Cap.

5  % S2.MinWorkingDays
6  working_day(C,D) :- assigned(C,D,P).
7  penalty("MinWorkingDays",course(C,MWD,N),(MWD-N)*weight_of_s2) :-
8          course(C,_,_,MWD,_,_), N = { working_day(C,D) }, N < MWD.

10 % S3.IsolatedLectures
11 scheduled_curricula(Cu,D,P) :- assigned(C,D,P), curricula(Cu,C).
12 penalty("IsolatedLectures",isolated_lectures(Cu,D,P),weight_of_s3) :-
13         scheduled_curricula(Cu,D,P), not scheduled_curricula(Cu,D,P-1),
14         not scheduled_curricula(Cu,D,P+1).

16 % S4.Windows
17 nscheduled_curricula(Cu,D,P) :- not scheduled_curricula(Cu,D,P), cu(Cu), ppd(P), d(D).
18 penalty("Windows",windows(Cu,C1,C2,D,P1,P2),(P2-P1-1)*weight_of_s4) :-
19         curricula(Cu,C1), curricula(Cu,C2), assigned(C1,D,P1), assigned(C2,D,P2),
20         P1 + 1 < P2, nscheduled_curricula(Cu,D,P) : P = P1+1..P2-1.

22 % S5.RoomStability
23 using_room(C,R) :- assigned(C,R,D,P).
24 penalty("RoomStability",using_room(C,N),(N-1)*weight_of_s5) :-
25         c(C), N = { using_room(C,R) }, N > 1.

27 % S6.StudentMinMaxLoad
28 penalty("StudentMinMaxLoad",student_min_max_load(Cu,D,N,many),(N-Max)*weight_of_s6) :-
29         cu(Cu), d(D), N = { assigned(C,D,P) : curricula(Cu,C), ppd(P) },
30         min_max_daily_lectures(Min,Max), N > Max.
31 penalty("StudentMinMaxLoad",student_min_max_load(Cu,D,N,few),(Min-N)*weight_of_s6) :-
32         cu(Cu), d(D), N = { assigned(C,D,P) : curricula(Cu,C), ppd(P) },
33         min_max_daily_lectures(Min,Max), 0 < N, N < Min.

35 % S7.TravelDistance
36 penalty("TravelDistance",instantaneous_move(Cu,C1,C2,D,P,P+1),weight_of_s7) :-
37         curricula(Cu,C1), curricula(Cu,C2), assigned(C1,R1,D,P), assigned(C2,R2,D,P+1),
38         room(R1,_,BLG1), room(R2,_,BLG2), BLG1 != BLG2.

40 % S8.RoomSuitability
41 penalty("RoomSuitability",assigned(C,R,D,P),weight_of_s8) :-
42         assigned(C,R,D,P), room_constraint(C,R).

44 % S9.DoubleLectures
45 penalty("DoubleLectures",non_grouped_lecture(C,R,D,P),weight_of_s9) :-
46         course(C,_,_,_,_,1), d(D), 2 { assigned(C,D,PPD) },
47         assigned(C,R,D,P), not assigned(C,R,D,P-1), not assigned(C,R,D,P+1).

49 % objective function
50 #minimize { P,C,S : penalty(S,C,P) }.
```

**Listing 5** Encoding of soft constraints and objective function

rule in Line 14 constrains that, for every room `R`, day `D`, and period `P`, there is at most one course `C` such that `assigned(C,R,D,P)` holds. For $H_4$, the rule in Line 17 constrains that, for every room `R`, a course `C` is not assigned to a room `R` at a period `P` on a day `D`, if `unavailability_constraint(C,D,P)` holds. The rule in Line 20 constrains that, for a given number of rooms `N`, and for every day `D` and period `P`, there are at most `N` lectures such that `assigned(C,D,P)` holds. This rule expresses an implied constraint and can be omitted, we keep it as an additional rule for performance improvement of some problem instances.

The *teaspoon* encoding of soft constraints ($S_1$–$S_9$) is shown in Listing 5. The predicate `penalty(`$S_i$`,`$V$`,`$C$`)` is intended to express that a constraint $S_i$ is violated by $V$ and its penalty

cost is $C$. Here again, each constraint $S_i$ is compactly expressed by either one or two rules in which the head is the form of `penalty(`$S_i$`,`$V$`,`$C$`)`, and a violation $V$ and its penalty cost $C$ are detected and calculated respectively in the body. That is, for each violation $V$ of $S_i$, the predicate `penalty(`$S_i$`,`$V$`,`$C$`)` is generated. We refer to an instance of `penalty/3` as a *penalty atom*. This idea can be used to express a wide variety of soft constraints such as those with constant cost and calculated cost. The constants denoted by `weight_of_*` indicate the weights associated with each soft constraint defined in Table 1.

We give the explanation of $S_1$–$S_3$ that compose the basic formulation UD1. For $S_1$, the rule in Line 2, for every course `C` that `N` students attend and room `R` that has a seating capacity of `Cap`, generates a penalty atom with the cost of the production of `N-Cap` and `weight_of_s1`, if `N > Cap` and `assigned(C,R,D,P)` holds. For $S_2$, the rule in Line 6 generates an atom `working_day(C,D)` for every course `C`, day `D`, and period `P` if `assigned(C,D,P)` holds. The atom `working_day(C,D)` expresses that a course `C` is given on a day `D`. The rule in Line 7–8, for every course `C` whose lectures must be spread into `MWD` days, generates a penalty atom with the cost of the production of `MWD-N` and `weight_of_s2`, if the number of days (`N`) in which a course `C` spread is less than `MWD`. For $S_3$, the rule in Line 11 generates an atom `scheduled_curricula(Cu,D,P)` for every curriculum `Cu`, course `C` that belongs to `Cu`, day `D`, and period `P` if `assigned(C,D,P)` holds. The atom `scheduled_curricula(Cu,D,P)` expresses that a curriculum `Cu` is scheduled at a period `P` on a day `D`. The rule in Line 12–14, for every curriculum `Cu`, day `D`, and period `P`, generates a penalty atom with the constant cost `weight_of_s3`, if a curriculum `Cu` is scheduled at a period `P` on a day `D`, but not at `P-1` and `P+1` within the same day `D`. In the ITC-2007 competition setting, we have a single objective function for minimizing the weighted sum of penalty costs, which is shown in Line 50.

## 5 The *teaspoon* System

As mentioned, the *teaspoon* system accepts a standard input format, viz. `ectt` [10]. For this, we implemented a simple converter that provides us with the resulting CB-CTT instance in *teaspoon*'s fact format. In turn, these facts are combined with the *teaspoon* encoding, which is subsequently solved by the ASP system *clingo* that returns an assignment representing a solution to the original CB-CTT instance.

Our empirical analysis considers all instances in different formulations (UD1–UD5), which are publicly available from the CB-CTT portal. The benchmark set `ITC-2007` consisting of 21 instances denoted by `comp*`, `DDS-2008` of 7 instances by `DDS*`, `Test` of 5 instances by `test*`, `Erlangen` of 6 instances by `erlangen*`, `EasyAcademy` of 12 instances by `EA*`, and `Udine` of 9 instances by `Udine*`. Among them, the instances of `Erlangen` are very large. For example, the instance `erlangen2012_2.ectt` consists of 850 courses, 132 rooms, 850 curricula, 7,780 unavailability constraints, and 45,603 room constraints. We ran them on a cluster of Linux machines equipped with dual Xeon E5520 quad-core 2.26 GHz processors and 48 GB RAM. We imposed a limit of 3 hours and 20GB. We used *clingo* 5 [5] for our experiments.

Since *clingo* utilizes a variety of techniques and parameters guiding the search, we explored several configurations. We focused on parameters concerning optimization and configurations from *clingo*'s portfolio. Preliminary benchmarks on the `ITC-2007` instances eliminated suboptimal configurations. Furthermore, configurations were only considered if they had so-called "unique solutions" on the whole benchmark set. A solution for a configuration is called unique if there is no other configuration that has a better objective value

[5] We used revision r10140 of the current development branch available at `http://potassco.sourceforge.net/`.

or proven optimality for the same value. One configuration was automatically determined by *piclasp* 1.2.1, a configurator for *clingo* based on *smac* [26]. The parameter space was restricted to optimization related parameters and portfolio configurations. The `ITC-2007` instances served as training set[6] and each solver run was limited to 600 seconds.

We determined the following 15 configurations: *BB0*, *BB0-HEU3-RST*, *BB2*, *BB2-TR*, *Dom5*, *USC1*, *USC11*, *USC11-CR*, *USC11-JP*, *USC13*, *USC13-CR*, *USC13-HEU3-RST-HD (LRND)*, *USC3-JP*, *USC15*, *USC15-CR* which consist of a variety of *clingo*'s search options:

- ***BB**n*: *Model-guided* branch-and-bound approach traditionally used for optimization in ASP [3]. The idea is to iteratively produce models of descending costs until the optimal is found by establishing unsatisfiability of finding a model with lower cost. Parameter *n* controls how the costs are step-wise reduced, either strict lexicographically, hierarchically, exponentially increasing or exponentially decreasing.
- ***USC**n*: *Core-guided* optimization techniques originated in MaxSAT [9]. Core-guided approaches rely on successively identifying and relaxing unsatisfiable cores until a model is obtained. The parameter *n* indicates what refinements and algorithms are used, e.g. algorithms *oll* [4], *pmres* [34], the combination of both with disjoint core preprocessing [31] and whether the constraints used to relax an unsatisfiable core are added as implications or equivalences. For $n > 8$, a technique called *stratification* [5] is enabled. *Stratification* refines lower bound improving algorithms on handling weighted instances. The idea is to focus at each iteration on soft constraints with higher weights by properly restricting the set of rules added to the solving process. The goal is to faster obtain a better bound without having to prove optimality.
- ***HEU3***: Enables optimization-oriented model and sign heuristic.
- ***RST***: The solver performs a restart after every intermediate model that was found.
- ***DOM5***: Atoms that are used in the optimization statement are preferred as decision variables in the solving algorithm and the sign heuristic tries to make those atoms true. The technique used to modify the variables is called *domain-specific heuristic* and is presented in [23].
- ***LRND***: Refers to the configuration automatically learned by *piclasp*. For space reasons, the configuration is refereed to as *LRND* from here on out.
- ***CR***: Refers to *clingo*'s configuration *crafty* that is geared towards crafted problems.
- ***HD***: Refers to *clingo*'s configuration *handy* that is geared towards larger problems.
- ***JP***: Refers to *clingo*'s configuration *jumpy* that uses more aggressive defaults.
- If neither ***CR***, ***JP*** or ***HD*** is specified, *clingo*'s default configuration for ASP problems *tweety* is taken. This configuration was determined by *piclasp* and refined manually. For more information on *clingo*'s search configurations, see [20].

We introduce the notion of *k-way configurations*. A *k*-way configuration is a set of *k* configurations, chosen from the 15 aforementioned configurations. The result of a *k*-way configuration for each instance is the best result among the *k* configurations in the set. For example, $\{USC1, BB0, USC11\}$ is a 3-way configuration with the best results between *USC1*, *BB0* and *USC11*. Intuitively, 1-way configurations are equal to the 15 configurations listed above and the only 15-way configuration is equal to the virtual best solver, referred to as *VBS-ASP*.

At first, we analyze the difference between the configurations. To this end, Table 2 contrasts the results obtained from *clingo*'s different configurations, the best *k*-way configurations where $2 \leq k \leq 14$, as well as the virtual best configuration *VBS-ASP*. The configura-

[6] We are aware that the training set is included in the test set. The decision was made since no separate instance set was available and we wanted to record results for all instances and configurations.

**Table 2** Comparison between different *clingo* configurations

| Configuration | Mean rank | #Optimal solutions | #Unsolved solutions | #Unique solutions |
|---|---|---|---|---|
| *VBS-ASP* | 12267.77 | 125 | 0 | - |
| Best 14-way configuration | 12349.69 | 125 | 0 | - |
| Best 13-way configuration | 12431.61 | 125 | 0 | - |
| Best 12-way configuration | 12574.97 | 125 | 0 | - |
| Best 11-way configuration | 12738.81 | 125 | 0 | - |
| Best 10-way configuration | 12981.16 | 125 | 0 | - |
| Best 9-way configuration | 13257.64 | 125 | 0 | - |
| Best 8-way configuration | 13564.84 | 125 | 0 | - |
| Best 7-way configuration | 13895.08 | 125 | 0 | - |
| Best 6-way configuration | 14533.37 | 125 | 0 | - |
| Best 5-way configuration | 15192.15 | 125 | 0 | - |
| Best 4-way configuration | 16418.60 | 122 | 0 | - |
| Best 3-way configuration | 17789.11 | 122 | 0 | - |
| Best 2-way configuration | 19595.98 | 122 | 0 | - |
| *USC11-JP* | 23288.33 | 122 | 0 | 21 |
| *USC11* | 23938.37 | 119 | 0 | 6 |
| *BB0-HEU3-RST* | 24056.47 | 77 | 0 | 23 |
| *USC13* | 24272.54 | 116 | 0 | 3 |
| *USC15* | 24280.83 | 117 | 0 | 2 |
| *USC13-CR* | 24318.72 | 116 | 10 | 5 |
| *USC15-CR* | 24346.08 | 118 | 2 | 4 |
| *USC11-CR* | 24381.20 | 116 | 12 | 3 |
| *USC13-HEU3-RST-HD* (*LRND*) | 24638.01 | 115 | 0 | 10 |
| *BB2-TR* | 25063.97 | 79 | 0 | 27 |
| *USC3-JP* | 25259.70 | 122 | 120 | 6 |
| *BB0* | 25367.61 | 73 | 0 | 14 |
| *USC1* | 25888.89 | 118 | 129 | 2 |
| *BB2* | 26740.59 | 78 | 0 | 6 |
| *Dom5* | 27384.24 | 76 | 155 | 7 |

tions are ordered by the mean rank that was calculated as suggested in the ITC 2007. [7] Since there was no distance to feasibility available, it was assumed to be the same for all configurations and instances. Table 2 also displays the number of optimal solutions, unsolved instances and unique solution for each configuration.

The highest-ranked single configuration was *USC11-JP* with also the highest number of optimal solutions among the single configurations, though the same number of optimal solutions was obtained by *USC3-JP*. Overall, *core-guided* strategies with *stratification* seem to provide a good trade-off between providing intermediate solutions with good upper bounds and proving optimality. The only *model-driven* configuration among the top single configuration is *BB0-HEU3-RST*. The optimization-tailored heuristics and frequent restarts seem to improve convergence of the objective function value, but do not help in proving optimality, since, despite its high rank, the configuration found the third least optimal solutions.

No smaller best $j$-way configuration was able to beat or be as good as a best $i$-way configuration where $j < i$. Adding more configurations continuously improves the mean rank and the total number of 125 optimal solutions is reached with combining five configurations. Since the mean rank takes into account the individual ranking of the objective value for each instance, the large distance in mean rank between the best single configuration and the best virtual configuration indicates that the different instances are sensitive to different configurations.

Table 3 shows which single configurations are included in the best $k$-way configurations. Each column represents one best $k$-way configuration and each row a single configuration. A $\times$ indicates that the configuration is included in the best $k$-way configuration in

[7] `http://www.cs.qub.ac.uk/itc2007/index_files/ordering.htm`

**Table 3** Best k-way configurations

| Configuration \ $k$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | #Included in best |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *BB0-HEU3-RST* | × | × | × | × | × | × | × | × | × | × | × | × | × | 13 |
| *USC11-JP* | × | × | × | × | × | × | × | × | × | × | × | × | × | 13 |
| *BB2-TR* | | × | × | × | × | × | × | × | × | × | × | × | × | 12 |
| *USC13-CR* | | | × | × | × | × | × | × | × | × | × | × | × | 11 |
| *USC11* | | | | × | × | × | × | × | × | × | × | × | × | 10 |
| *BB0* | | | | | × | × | × | × | × | × | × | × | × | 9 |
| *LRND* | | | | | | × | × | × | × | × | × | × | × | 8 |
| *USC3-JP* | | | | | | | × | × | × | × | × | × | × | 7 |
| *Dom5* | | | | | | | | × | × | × | × | × | × | 6 |
| *BB2* | | | | | | | | | × | × | × | × | × | 5 |
| *USC15-CR* | | | | | | | | | | × | × | × | × | 4 |
| *USC13* | | | | | | | | | | | × | × | × | 3 |
| *USC11-CR* | | | | | | | | | | | | × | × | 2 |
| *USC15* | | | | | | | | | | | | | × | 1 |

that row. The last column shows how many times the configuration in that row was in a best $k$-way configuration. The only single configuration that is not included is *USC1* since it was not in any best $k$-way configuration. For example, the best 5-way configuration is $\{$*BB0-HEU3-RST*, *USC11-JP*, *BB2-TR*, *USC13-CR*, *USC11*$\}$.

All best $k$-way configurations are contained in best $k+1$-way configurations for all $2 \leq k \leq 13$ in the table. So increasing $k$ boils down to adding a configuration that provides upper bounds improving the ranking in an optimal way. *USC11-JP* and *BB0-HEU3-RST* are included in all best $k$-way configurations. This correlates with the individual ranking of the single configurations, where *USC11-JP* placed first and *BB0-HEU3-RST* third respectively. However, the next configuration added, viz. *BB2-TR*, has the most unique solutions but is individually ranked 10th. Unique solutions provide a definite improvement of the mean rank, because it is guaranteed to improve the rank of at least a number of instances equal to the number of unique solutions. Though, the correlation of the order of configurations added and number of unique solutions is not exact. A new configuration that adds upper bounds for an instance that tie for first place also improve the overall mean rank. Other examples of this observation are *Dom5*, ranked last but included in 6 best $k$-way configurations, and *USC15*, ranked 5th but only in one best $k$-way configuration. *Dom5* has seven and *USC15* two unique solutions.

Information about the best $k$-way configurations can be used to optimally configure a multi-threaded portfolio configuration whenever $k$ threads are available. The results show that each instance is configuration-sensitive, and combining configurations in an optimal way improves the results significantly.

In *VBS-ASP*, the time in seconds of finding optimal solutions for each combination of instance and formulation is shown in Table 4. After the individual times for each formulation, the next row shows the number of optimal solutions and the average time for the preceding formulation. The table below shows the overall number of optimal solution and the average time for all combinations. The overall average of 225.82 seconds is low compared to the time limit of 3 hours, the highest time for a combination being approximately one hour and 9 minutes. With increasing formulation number, the number of optimal solutions decreases and, except for UD4, the average time increases.

Next, we compare the performance of *teaspoon* with other approaches. Table 5 contrasts the best results of *teaspoon* with the best known ones on the CB-CTT web portal [8]. The

[8] `http://tabu.diegm.uniud.it/ctt/` on 2015-11-13

**Table 4** *VBS-ASP* : the times of finding optimal solutions

| Instance | Formulation | Time (sec.) |
|---|---|---|
| comp02 | UD1 | 2191.82 |
| comp04 | UD1 | 1.00 |
| comp06 | UD1 | 24.43 |
| comp07 | UD1 | 9.66 |
| comp08 | UD1 | 1.48 |
| comp10 | UD1 | 1.43 |
| comp11 | UD1 | 0.24 |
| comp13 | UD1 | 8.72 |
| comp14 | UD1 | 8.90 |
| comp16 | UD1 | 3.30 |
| comp17 | UD1 | 76.88 |
| comp19 | UD1 | 5.72 |
| comp20 | UD1 | 90.86 |
| DDS1 | UD1 | 953.51 |
| DDS2 | UD1 | 0.37 |
| DDS3 | UD1 | 0.19 |
| DDS5 | UD1 | 1.34 |
| DDS6 | UD1 | 1.78 |
| DDS7 | UD1 | 0.27 |
| EA01 | UD1 | 1.90 |
| EA02 | UD1 | 0.45 |
| EA04 | UD1 | 2.60 |
| EA05 | UD1 | 1.60 |
| EA06 | UD1 | 0.51 |
| EA07 | UD1 | 4.23 |
| EA08 | UD1 | 1.42 |
| EA09 | UD1 | 1.51 |
| EA10 | UD1 | 0.39 |
| EA11 | UD1 | 0.38 |
| EA12 | UD1 | 0.48 |
| test2 | UD1 | 1.89 |
| test3 | UD1 | 9.95 |
| toy | UD1 | 0.01 |
| Udine1 | UD1 | 3.44 |
| Udine2 | UD1 | 1.89 |
| Udine3 | UD1 | 1.71 |
| Udine4 | UD1 | 5.02 |
| Udine5 | UD1 | 1.61 |
| Udine6 | UD1 | 1.08 |
| Udine7 | UD1 | 1.0 |
| Udine8 | UD1 | 589.89 |
| Udine9 | UD1 | 2.60 |
| #42 | UD1 | 95.66 |

| Instance | Formulation | Time (sec.) |
|---|---|---|
| comp02 | UD2 | 5457.97 |
| comp04 | UD2 | 1.89 |
| comp06 | UD2 | 113.57 |
| comp07 | UD2 | 369.27 |
| comp08 | UD2 | 2.56 |
| comp10 | UD2 | 29.17 |
| comp11 | UD2 | 0.41 |
| comp13 | UD2 | 21.25 |
| comp14 | UD2 | 15.91 |
| comp16 | UD2 | 8.24 |
| comp17 | UD2 | 515.23 |
| comp19 | UD2 | 26.46 |
| comp20 | UD2 | 92.86 |
| DDS1 | UD2 | 230.39 |
| DDS2 | UD2 | 0.43 |
| DDS3 | UD2 | 0.24 |
| DDS5 | UD2 | 1.81 |
| DDS6 | UD2 | 13.98 |
| DDS7 | UD2 | 0.35 |
| EA01 | UD2 | 2.04 |
| EA02 | UD2 | 0.56 |
| EA04 | UD2 | 2.99 |
| EA05 | UD2 | 1.45 |
| EA06 | UD2 | 0.59 |
| EA07 | UD2 | 4.98 |
| EA08 | UD2 | 1.64 |
| EA09 | UD2 | 2.23 |
| EA10 | UD2 | 0.58 |
| EA11 | UD2 | 0.34 |
| EA12 | UD2 | 0.59 |
| test2 | UD2 | 8.37 |
| toy | UD2 | 0.01 |
| Udine1 | UD2 | 3.24 |
| Udine2 | UD2 | 3.21 |
| Udine3 | UD2 | 5.41 |
| Udine4 | UD2 | 3.56 |
| Udine5 | UD2 | 2.00 |
| Udine6 | UD2 | 1.26 |
| Udine7 | UD2 | 1.64 |
| Udine8 | UD2 | 372.09 |
| Udine9 | UD2 | 2.81 |
| #41 | UD2 | 178.62 |

| Instance | Formulation | Time (sec.) |
|---|---|---|
| comp02 | UD3 | 4123.27 |
| comp04 | UD3 | 1.19 |
| comp06 | UD3 | 2.98 |
| comp07 | UD3 | 16.11 |
| comp08 | UD3 | 5.67 |
| comp09 | UD3 | 267.69 |
| comp10 | UD3 | 1.98 |
| comp11 | UD3 | 0.44 |
| comp14 | UD3 | 1.39 |
| comp16 | UD3 | 1.76 |
| comp17 | UD3 | 2.12 |
| comp18 | UD3 | 2.12 |
| comp20 | UD3 | 137.79 |
| DDS6 | UD3 | 1.52 |
| test2 | UD3 | 0.35 |
| test3 | UD3 | 0.58 |
| test4 | UD3 | 3827.19 |
| toy | UD3 | 0.02 |
| Udine4 | UD3 | 15.65 |
| #19 | UD3 | 442.54 |
| comp04 | UD4 | 2.12 |
| comp06 | UD4 | 48.63 |
| comp07 | UD4 | 38.93 |
| comp08 | UD4 | 15.87 |
| comp10 | UD4 | 7.51 |
| comp11 | UD4 | 0.92 |
| comp14 | UD4 | 7.51 |
| comp16 | UD4 | 5.85 |
| comp17 | UD4 | 688.32 |
| comp20 | UD4 | 3388.42 |
| DDS6 | UD4 | 3.17 |
| test2 | UD4 | 1.46 |
| test3 | UD4 | 2.49 |
| toy | UD4 | 0.02 |
| Udine4 | UD4 | 27.69 |
| #15 | UD4 | 282.59 |
| comp04 | UD5 | 32.62 |
| comp08 | UD5 | 77.70 |
| comp11 | UD5 | 9.55 |
| DDS5 | UD5 | 137.60 |
| test2 | UD5 | 2365.18 |
| test3 | UD5 | 1142.12 |
| toy | UD5 | 0.02 |
| Udine4 | UD5 | 140.63 |
| #8 | UD5 | 488.18 |

| #Optimal solutions | Average time (sec.) |
|---|---|
| #125 | 225.82 |

symbols '$>$' and '$=$' indicate that *teaspoon* produced the improved and the same bounds respectively, compared to the previous best known bounds. If followed by a superscript '$*$', these symbols indicate that *teaspoon* proved the optimality of the obtained bounds. That is, the symbol "$>^*$" indicates that we found and proved a new optimal solution. The symbol '*n.a*' indicates that the result was not available on the web before.

We succeeded either in improving the bounds or producing the same bounds for 166 combinations (55,3% in the total), compared with the previous best known bounds. More precisely, the *teaspoon* encoding was able to improve the bounds for 43 combinations and to prove that 14 of them are optimal. That is, we found and proved new optimal solutions for 14 combinations. It was also able to produce the same bounds for 123 combinations and to prove for the first time that 35 of them are optimal. Furthermore, *teaspoon* was able to

**Table 5** Comparison of *teaspoon* with other approaches

| Instance | UD1 | | | UD2 | | | UD3 | | | UD4 | | | UD5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best known | | *VBS -ASP* | Best known | | *VBS -ASP* | Best known | | *VBS -ASP* | Best known | | *VBS -ASP* | Best known | | *VBS -ASP* |
| comp01 | 4 | = | 4 | 5 | = | 5 | 8 | = | 8 | 6 | | 9 | 11 | | 72 |
| comp02 | 12 | $=^*$ | 12 | 24 | $=^*$ | 24 | 12 | $=^*$ | 12 | 26 | | 55 | 130 | | 338 |
| comp03 | 38 | | 53 | 64 | | 109 | 25 | | 47 | 362 | | 405 | 142 | | 238 |
| comp04 | 18 | = | 18 | 35 | = | 35 | 2 | $=^*$ | 2 | 13 | $=^*$ | 13 | 59 | $>^*$ | 49 |
| comp05 | 219 | | 504 | 284 | | 624 | 264 | | 556 | 260 | | 459 | 570 | | 1081 |
| comp06 | 14 | $=^*$ | 14 | 27 | = | 27 | 8 | $=^*$ | 8 | 15 | $>^*$ | 9 | 85 | | 819 |
| comp07 | 3 | = | 3 | 6 | = | 6 | 0 | = | 0 | 3 | $=^*$ | 3 | 42 | | 962 |
| comp08 | 19 | = | 19 | 37 | = | 37 | 2 | $=^*$ | 2 | 15 | $=^*$ | 15 | 62 | $>^*$ | 55 |
| comp09 | 54 | | 63 | 96 | | 169 | 8 | $=^*$ | 8 | 38 | | 50 | 150 | | 215 |
| comp10 | 2 | = | 2 | 4 | = | 4 | 0 | = | 0 | 3 | $=^*$ | 3 | 72 | | 591 |
| comp11 | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 |
| comp12 | 239 | | 343 | 294 | | 456 | 51 | | 114 | 99 | | 388 | 483 | | 1135 |
| comp13 | 32 | $>^*$ | 31 | 59 | = | 59 | 22 | | 50 | 41 | | 111 | 148 | | 276 |
| comp14 | 27 | $=^*$ | 27 | 51 | = | 51 | 0 | = | 0 | 16 | $>^*$ | 14 | 95 | | 311 |
| comp15 | 38 | | 53 | 62 | | 109 | 16 | | 22 | 30 | | 68 | 176 | | 379 |
| comp16 | 11 | $=^*$ | 11 | 18 | = | 18 | 4 | $=^*$ | 4 | 7 | $=^*$ | 7 | 96 | | 906 |
| comp17 | 30 | $=^*$ | 30 | 56 | = | 56 | 12 | $=^*$ | 12 | 26 | $>^*$ | 21 | 155 | | 391 |
| comp18 | 34 | | 48 | 61 | | 81 | 0 | = | 0 | 27 | | 46 | 137 | | 228 |
| comp19 | 32 | $>^*$ | 29 | 57 | = | 57 | 24 | | 32 | 32 | | 82 | 125 | | 286 |
| comp20 | 2 | = | 2 | 4 | = | 4 | 0 | = | 0 | 9 | $>^*$ | 3 | 124 | | 1098 |
| comp21 | 43 | | 94 | 74 | | 124 | 6 | = | 6 | 36 | | 76 | 151 | | 215 |
| DDS1 | 38 | $=^*$ | 38 | 48 | = | 48 | 2393 | | 6036 | 2278 | = | 2278 | 1831 | | 5976 |
| DDS2 | 0 | = | 0 | 0 | = | 0 | 120 | | 379 | 76 | | 139 | 64 | | 212 |
| DDS3 | 0 | = | 0 | 0 | = | 0 | 22 | = | 22 | 11 | = | 11 | 22 | = | 22 |
| DDS4 | 16 | | 19 | 17 | | 33 | 54 | | 912 | 124 | | 1825 | 96 | | 2384 |
| DDS5 | 0 | = | 0 | 0 | = | 0 | 54 | | 117 | 163 | | 488 | 88 | $>^*$ | 76 |
| DDS6 | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 | 96 | | 864 |
| DDS7 | 0 | = | 0 | 0 | = | 0 | 30 | | 408 | 21 | | 506 | 52 | | 786 |
| EA01 | 55 | $=^*$ | 55 | 65 | $=^*$ | 65 | 102 | | 110 | 67 | | 88 | 196 | | 645 |
| EA02 | 0 | = | 0 | 0 | = | 0 | 96 | | 263 | 41 | | 262 | 128 | | 492 |
| EA03 | 1 | = | 1 | 2 | = | 2 | 50 | | 234 | 6936 | > | 816 | 90 | | 1750 |
| EA04 | 0 | = | 0 | 0 | = | 0 | 18 | | 21 | 9 | | 695 | 18 | | 99 |
| EA05 | 0 | = | 0 | 0 | = | 0 | 14 | = | 14 | 7 | | 8 | 14 | | 61 |
| EA06 | 5 | $=^*$ | 5 | 5 | $=^*$ | 5 | 42 | | 156 | 27 | | 336 | 99 | | 581 |
| EA07 | 0 | = | 0 | 0 | = | 0 | 206 | | 1822 | 3884 | > | 1122 | 205 | | 1681 |
| EA08 | 0 | = | 0 | 0 | = | 0 | 40 | | 48 | 20 | | 82 | 40 | | 181 |
| EA09 | 2 | $=^*$ | 2 | 4 | $=^*$ | 4 | 40 | = | 40 | 22 | | 27 | 48 | | 100 |
| EA10 | 0 | = | 0 | 0 | = | 0 | 4 | | 141 | 19 | | 573 | 93 | | 544 |
| EA11 | 0 | = | 0 | 0 | = | 0 | 36 | | 52 | 19 | | 22 | 45 | | 93 |
| EA12 | 2 | $=^*$ | 2 | 4 | $=^*$ | 4 | 22 | | 38 | 12 | | 24 | 27 | | 126 |
| erlangen2011_2 | 3061 | > | 1662 | 4670 | | 5733 | 8122 | > | 7260 | 3152 | > | 2816 | 8010 | > | 7892 |
| erlangen2012_1 | 2782 | > | 2631 | 5716 | > | 2928 | 7544 | > | 3574 | 2694 | > | 2176 | 7585 | > | 6702 |
| erlangen2012_2 | 3332 | > | 3324 | 8813 | > | 6818 | 9731 | > | 4037 | 4624 | > | 3930 | 9081 | > | 7874 |
| erlangen2013_1 | 2608 | > | 1372 | 5476 | > | 3553 | 7289 | > | 3358 | 3553 | > | 3099 | 7253 | > | 6543 |
| erlangen2013_2 | *n.a* | > | 9901 | 8150 | | 19839 | *n.a* | > | 23285 | *n.a* | > | 12682 | *n.a* | > | 26621 |
| erlangen2014_1 | *n.a* | > | 9497 | 5981 | | 18395 | *n.a* | > | 20286 | *n.a* | > | 8048 | *n.a* | > | 22376 |
| test1 | 212 | | 328 | 224 | | 404 | 200 | | 299 | 208 | | 413 | 232 | | 539 |
| test2 | 8 | = | 8 | 16 | = | 16 | 0 | = | 0 | 4 | $=^*$ | 4 | 20 | $=^*$ | 20 |
| test3 | 35 | = | 35 | 67 | | 113 | 18 | $=^*$ | 18 | 18 | $>^*$ | 17 | 97 | $>^*$ | 68 |
| test4 | 27 | | 91 | 73 | | 156 | 12 | $>^*$ | 6 | 33 | | 37 | 166 | | 401 |
| toy | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 | 0 | = | 0 |
| Udine1 | 0 | = | 0 | 0 | = | 0 | 128 | | 426 | 64 | | 427 | 138 | | 420 |
| Udine2 | 4 | = | 4 | 8 | $=^*$ | 8 | 34 | | 322 | 30 | | 320 | 81 | | 313 |
| Udine3 | 0 | = | 0 | 0 | = | 0 | 24 | | 88 | 19 | | 67 | 54 | | 175 |
| Udine4 | 35 | = | 35 | 64 | = | 64 | 24 | $=^*$ | 24 | 31 | $=^*$ | 31 | 108 | $>^*$ | 106 |
| Udine5 | 0 | = | 0 | 0 | = | 0 | 44 | | 338 | 23 | | 145 | 47 | | 294 |
| Udine6 | 0 | = | 0 | 0 | = | 0 | 36 | | 76 | 18 | | 50 | 38 | | 111 |
| Udine7 | 0 | = | 0 | 0 | = | 0 | 64 | | 94 | 32 | | 62 | 64 | | 116 |
| Udine8 | 16 | $=^*$ | 16 | 31 | $>^*$ | 29 | 42 | | 297 | 31 | | 149 | 88 | | 172 |
| Udine9 | 18 | = | 18 | 21 | $=^*$ | 21 | 28 | | 62 | 23 | | 91 | 70 | | 163 |

produce upper bounds for very large instances in the category `erlangen` with every formulation, and 8 of them were unsolvable before.

Finally, we briefly compare the new results with our previous work [6]. The 185 benchmark instances in [6] were a subset of the CB-CTT portal benchmark set, comprised of the 5 formulations for the categories `comp`, `DDS`, `test` and `erlangen` without `erlangen2013_2` and `erlangen2014_1`. The *teaspoon* system was able to obtain the same or better bounds for 168 combinations (90.8% in the total). In detail, *teaspoon* improves bounds for 109 combinations and proves optimality for 30 of them. For 59 combinations, the same bounds were produced and 6 of them were confirmed to be optimal.

## 6 Extensions

We here extend the basic *teaspoon* encoding presented in Section 4 in view of enhancing the scalability and flexibility of solving (multi-criteria) CB-CTT problems.

**Optimized encodings for the soft constraints $S_7$, $S_4$, $S_2$, and $S_6$.** The basic encoding of $S_7$ and $S_4$ precisely reflects their definition but fails to scale to large instances in complex formulations like UD5 due to expensive grounding. For $S_7$, the rule in Line 36–38 of Listing 5 generates a penalty atom with the constant cost `weight_of_s7` if both `assigned(C1,R1,D,P)` and `assigned(C2,R2,D,P+1)` hold for two courses `C1` and `C2` that belong to some curriculum `Cu`, day `D`, and period `P`, and rooms `R1` and `R2` are located in different buildings. This rule is expensive when grounding due to its combinatorial blow-up caused by many variables. This issue can be improved by taking into account that for every curriculum `Cu`, room `R`, day `D`, and period `P`, there is at most one course `C` that belongs to `Cu` such that `assigned(C,R,D,P)` holds. In view of this, an optimized encoding of $S_7$ is shown in Line 2–4 of Listing 6. The difference from the basic one is that a new predicate `scheduled_curricula/4` is introduced. The atom `scheduled_curricula(Cu,B,D,P)` is intended to express that a curriculum `Cu` is scheduled in a building `B` at a period `P` on a day `D`. The rule in Line 2 generates an atom `scheduled_curricula(Cu,B,D,P)` if `assigned(C,R,D,P)` holds for every curriculum `Cu`, course `C` that belongs to `Cu`, room `R` located in a building `B`, day `D`, and period `P`. The rule in Line 3–4 produces a penalty atom with the constant cost `weight_of_s7` for every curriculum `Cu`, day `D`, and period `P`, if a curriculum `Cu` is scheduled in different buildings at period `P` and `P+1` within the same day `D`. Another optimized encoding of $S_7$ is shown in Line 7–9 of Listing 6. The difference from the other two is that it utilizes cardinality constraints for counting the number of buildings which are used by two lectures belonging the same curriculum in two adjacent periods within the same day.

An optimized encoding of $S_4$ is shown in Line 12–19 of Listing 6. The newly introduced atom `scheduled_curricula_chain(Cu,D,P,DP)` is intended to express that there is a course in curriculum `Cu` scheduled before a period `P` in a day `D` if `DP = -1`, or else if `DP = 1` the course in `Cu` is scheduled after `P`. The rule in Line 16–19 generates a penalty atom with the constant cost `weight_of_s4` for every curriculum `Cu`, day `D`, and period `P`, if there is a time window `P` for `Cu` in a day `D`.

In the basic encoding, the soft constraints $S_2$ and $S_6$ are expressed by using ASP's cardinality constraints. These rules can be optimized by using state-of-the-art SAT encoding techniques for Boolean cardinality constraints. We used Sinz's sequential counter encoding [37], and the resulting encodings are shown in Line 22–27 for $S_2$ and Line 30–45 for $S_6$. For $S_2$, the atom `wd_counter(`$C$`,`$M$`,`$D$`,`$N$`)` is intended to express that the number of lectures scheduled from day 0 to $D$ for a course $C$ whose lectures must be spread into $M$ days

```
1  % S7.TravelDistance
2  scheduled_curricula(Cu,B,D,P) :- assigned(C,R,D,P), curricula(Cu,C), room(R,_,B).
3  penalty("TravelDistance",instantaneous_move(Cu,D,P,P+1),weight_of_s7) :-
4      scheduled_curricula(Cu,BLG1,D,P), scheduled_curricula(Cu,BLG2,D,P+1), BLG1 != BLG2.

6  % S7.TravelDistance
7  penalty("TravelDistance",instantaneous_move(Cu,D,P,P+1),weight_of_s7) :-
8      cu(Cu), d(D), ppd(P), ppd(P+1),
9      #count { B : assigned(C,R,D,(P;P+1)), curricula(Cu,C), room(R,_,B) } > 1.

11 % S4.Windows
12 dp(1;-1).
13 scheduled_curricula(Cu,D,P) :- assigned(C,D,P), curricula(Cu,C).
14 scheduled_curricula_chain(Cu,D,P,   DP) :- scheduled_curricula(Cu,D,P), ppd(P+DP), dp(DP).
15 scheduled_curricula_chain(Cu,D,P+DP,DP) :- scheduled_curricula_chain(Cu,D,P,DP), ppd(P+DP).
16 penalty("Windows",windows(Cu,D,P),weight_of_s4) :-
17     scheduled_curricula_chain(Cu,D,P,-1),
18     not scheduled_curricula(Cu,D,P),
19     scheduled_curricula_chain(Cu,D,P,1).

21 % S2.MinWorkingDays
22 working_day(C,D) :- assigned(C,D,P).
23 wd_counter(C,M,-1,0) :- course(C,_,_,M,_,_).
24 wd_counter(C,M,D,N+1) :- wd_counter(C,M,D-1,N), working_day(C,D), N+1 <= M.
25 wd_counter(C,M,D,N+0) :- wd_counter(C,M,D-1,N), d(D), N <= M.
26 penalty("MinWorkingDays",course(C,N),weight_of_s2) :-
27     course(C,_,_,M,_,_), N = 1..M, days(D), not wd_counter(C,M,D-1,N).

29 % S6.StudentMinMaxLoad
30 abc(M,min)  :- min_max_daily_lectures(M,_).
31 abc(M,max)  :- min_max_daily_lectures(_,Max), periods_per_day(Ppd), M=Ppd-Max.
32 abc(Cu,D,P) :- assigned(C,D,P), curricula(Cu,C).
33 abc_counter(Cu,D,-1,  0,min) :- cu(Cu), d(D).
34 abc_counter(Cu,D,-1,  0,max) :- cu(Cu), d(D).
35 abc_counter(Cu,D, P,N+1,min) :-
36     abc_counter(Cu,D,P-1,N,min),             abc(Cu,D,P), N+1 <= M, abc(M,min).
37 abc_counter(Cu,D, P,N+1,max) :-
38     abc_counter(Cu,D,P-1,N,max), ppd(P), not abc(Cu,D,P), N+1 <= M, abc(M,max).
39 abc_counter(Cu,D, P,N+0,MM)  :-
40     abc_counter(Cu,D,P-1,N,MM),  ppd(P),                  N   <= M, abc(M,MM).
41 abc_counter(Cu,D,min) :- abc(Cu,D,P).
42 abc_counter(Cu,D,max) :- cu(Cu), d(D).
43 penalty("StudentMinMaxLoad",student_min_max_load(Cu,D,N),weight_of_s6) :-
44     cu(Cu), d(D), N = 1..M, periods_per_day(P), abc(M,MM), abc_counter(Cu,D,MM),
45     not abc_counter(Cu,D,P-1,N,MM).
```

**Listing 6** A collection of optimized encodings for $S_7$, $S_4$, $S_2$, and $S_6$

```
1  % S8 RoomSuitability
2  penalty("RoomSuitability",assigned(C,R,D,P),W) :-
3          assigned(C,R,D,P), room_constraint(C,R), soft_constraint("RoomSuitability",W).
4  :- assigned(C,R,D,P), room_constraint(C,R), hard_constraint("RoomSuitability").
```

**Listing 7** Extended encoding of $S_8$

is greater than and equal to $N$. The rule in Line 26–27 generates a penalty atom with the constant cost `weight_of_s2` for every course `C` whose lectures must be spread into `M` days, if the number of lectures for `C` scheduled in the whole days is less than `M`.

**Easy composition of different formulations.** To easily activate or deactivate each soft constraint and switch it from soft to hard, we introduce new predicates `soft_constraint/2` and `hard_constraint/1`. The atom `soft_constraint(`$S_i$`,`$W_i$`)` is intended to express that $S_i$

```
soft_constraint("RoomCapacity",     1). soft_constraint("MinWorkingDays",   1).
soft_constraint("Windows",          1). soft_constraint("StudentMinMaxLoad",1).
hard_constraint("RoomSuitability"). soft_constraint("DoubleLectures",   1).
```

**Listing 8** The UD4 formulation

```
soft_constraint("RoomCapacity",      1). soft_constraint("MinWorkingDays",    1).
soft_constraint("IsolatedLectures",  1). soft_constraint("Windows",           1).
soft_constraint("RoomStability",     1). soft_constraint("StudentMinMaxLoad", 1).
soft_constraint("TravelDistance",    1). soft_constraint("RoomSuitability",   1).
soft_constraint("DoubleLectures",    1).
```

**Listing 9** Formulation consisting of all soft constraints ($S_1$–$S_9$) with the weights of all 1s

```
#minimize { P@9,C : penalty("RoomCapacity",C,P) }.
#minimize { P@8,C : penalty("MinWorkingDays",C,P) }.
#minimize { P@7,C : penalty("IsolatedLectures",C,P) }.
#minimize { P@6,C : penalty("Windows",C,P) }.
#minimize { P@5,C : penalty("RoomStability",C,P) }.
#minimize { P@4,C : penalty("StudentMinMaxLoad",C,P) }.
#minimize { P@3,C : penalty("TravelDistance",C,P) }.
#minimize { P@2,C : penalty("RoomSuitability",C,P) }.
#minimize { P@1,C : penalty("DoubleLectures",C,P) }.
```

**Listing 10** Multiple objective functions

is a soft constraint to be activated and its weight is $W_i$. The atom `hard_constraint(`$S_i$`)` is intended to express that $S_i$ is activated as a hard constraint. We refer to these atoms as *constraint atoms*. An extended encoding of $S_8$ with constraint atoms is shown in Listing 7. The rule in Line 2–3 is the same as before except for the instance of `soft_constraint/2`. The rule in Line 4 expresses $S_8$ as a hard constraint by dropping the penalty atom in the head. Another method for switching from soft to hard is to enforce a cardinality constraint like '`:- not #sum { P,C : penalty("RoomSuitability",C,P) } 0.`'. In this case, we don't need the rule in Line 4 that expresses $S_8$ as a hard constraint. It is also possible to switch constraints in the opposite direction. For example, to define $H_4$ as a soft constraint, we only have to add a penalty atom to the head of the rule in Line 17 of Listing 4.

The idea of constraint atoms allows for easy composition of different formulations, since any combination of constraints can be represented as a set of facts. Consequently, it enables a timetable keeper to experiment with different formulations at a purely declarative level. For example, ASP facts representing the UD4 formulation is shown in Listing 8. And also, we show a big formulation consisting of all soft constraints ($S_1$–$S_9$) with the weights of all 1s in Listing 9.

**Support for multi-criteria optimization based on lexicographic ordering.** A well-known multi-criteria optimization strategy called lexicographic ordering [30] has been implemented in *clingo*. It enables us to optimize criteria in a lexicographic order based on their priorities. We extend our basic encoding for supporting such multi-criteria optimization. This extension can be done by replacing the `#minimize` function in Listing 5 with multiple ones in Listing 10. Each integer value on the right-hand side of `@` stands for a priority level, where greater levels are more significant than smaller ones[9]. Usually, solutions can be rep-

[9] Note that the priorities in Listing 10 are quite arbitrary.

resented in the form of utility vector $(p_1, p_2, \ldots, p_n)$, where each $p_i$ stands for the penalty cost of soft constraint $S_i$.

On the other hand, the optimality of multi-criteria optimization with lexicographic ordering does not always coincide with that of single-objective one. However, optimal solutions obtained by lexicographic optimization can correspond to feasible ones with small penalty cost in the original single-objective setting. In a preliminary experiment on the UD5 formulation, the lexicographic optimization in the order of $S_1 > S_4 > S_2 > S_7 > S_6 > S_3$ found optimal solutions for more than half instances of ITC-2007 competition. The results include optimal vectors $(S_1, S_4, S_2, S_7, S_6, S_3) = (0,0,0,0,22,11)$ for `comp10`,$(0,0,0,0,112,35)$ for `comp13`,$(0,0,0,0,56,28)$ for `comp14`,$(0,0,20,0,122,54)$ for `comp09`, and $(0,0,10,0,110,46)$ for `comp21`. From single objective's point of view, the first three correspond to better bounds (33, 147, and 84) than the previous best known ones (72, 148, and 95). The last two correspond to bounds (196 and 166) which are close to the best known ones (150 and 151).

**Towards multi-shot ASP solving with *teaspoon*.** Incremental SAT solving has recently been recognized as an important technique for many problems such as model checking and planning [18]. From an ASP perspective, multi-shot ASP solving has been implemented in *clingo* [22]. It enables us to handle problem specifications which evolve during the reasoning process, either because data or constraints are added, deleted, or replaced.

For (multi-criteria) CB-CTT solving, multi-shot ASP solving with *teaspoon* can be promising. This is because it allows for incremental solving of finding optimal solutions with varying a set of constraints, switching them from hard to soft, varying the priority level of objectives, and reusing legacy timetables. Suppose that a legacy timetable is represented as a set of instances of predicate `legacy/1`. The reuse of a legacy timetable can be expressed by only one rule:

```
_heuristic(assigned(C,R,D,P),true,1) :- legacy(assigned(C,R,D,P)).
```

The special predicate `_heuristic/3` is used to express various modifications to the *clingo*'s heuristic treatment of atoms. This rule expresses a preference for both making a decision on `assigned(C,R,D,P)` and assigning it to true if `legacy(assigned(C,R,D,P))` holds for every course `C`, room `R`, day `D`, and period `P`. In a preliminary experiment with UD5, we were finally able to find new bounds 33 for `comp10`, 135 for `comp13`, 74 for `comp14`, 142 for `comp09`, and 143 for `comp21` by reusing the solutions obtained by lexicographic optimization as legacy timetables.

## 7 Discussion

Perhaps the most relevant works are problem encodings in Integer Programming [11–13, 27]. These encodings use the binary variables $x_{C,D,P}$ and/or $x_{C,R,D,P}$ that correspond to the predicate `assigned/3` and/or `assigned/4` respectively. SAT/MaxSAT encodings [2] also use the same binary variables. The major advantage of our approach is not only the compact and flexible declarative representation gained by using ASP as a modeling language, but also the high performance gained from the recent advanced techniques in ASP solving.

We presented an ASP-based approach for solving the CB-CTT problems. The resulting system *teaspoon* [10] relies on high-level ASP encodings and delegates both the grounding and solving tasks to general-purpose ASP systems. Our empirical analysis showed that core-guided optimization with stratification is very effective in finding optimal solutions. Further-

[10] All source code is available from `http://potassco.sourceforge.net/apps.html`

more, it also showed that each CB-CTT instance is configuration-sensitive, and combining configurations in an optimal way improves the performance significantly.

We have contrasted the performance of *teaspoon* with the best known bounds obtained so far via dedicated implementations. *teaspoon* demonstrated that ASP's general-purpose technology allows to compete with state-of-the-art CB-CTT solving techniques. In fact, the high-level approach of ASP facilitates extensions and variations of first-order encodings in view of enhancing the scalability and flexibility of solving (multi-criteria) CB-CTT problems. In the future, we thus aim at investigating multi-shot ASP solving with *teaspoon*.

## References


1. Abdullah, S., Turabieh, H., McCollum, B., McMullan, P.: A hybrid metaheuristic approach to the university course timetabling problem. Journal of Heuristics **18**(1), 1–23 (2012)
2. Achá, R.A., Nieuwenhuis, R.: Curriculum-based course timetabling with SAT and MaxSAT. Annals of Operations Research pp. 1–21 (2012)
3. Alviano, M., Dodaro, C., Marques-Silva, J., Ricca, F.: On the implementation of weak constraints in wasp. In: D. Inclezan, M. Maratea (eds.) Proceedings of the Seventh Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'14) (2014). URL https://sites.google.com/site/aspocp2014/paper_9.pdf
4. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: A. Dovier, V. Santos Costa (eds.) Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12), vol. 17, pp. 212–221. Leibniz International Proceedings in Informatics (LIPIcs) (2012)
5. Ansótegui, C., Bonet, M., Levy, J.: Sat-based maxsat algorithms. Artificial Intelligence **196**, 77–105 (2013)
6. Banbara, M., Soh, T., Tamura, N., Inoue, K., Schaub, T.: Answer set programming as a modeling language for course timetabling. Theory and Practice of Logic Programming **13**(4-5), 783–798 (2013)
7. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
8. Bettinelli, A., Cacchiani, V., Roberti, R., Paolo, Toth: An overview of curriculum-based course timetabling. TOP **23**(2), 313–349 (2015)
9. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, *Frontiers in Artificial Intelligence and Applications*, vol. 185. IOS Press (2009)
10. Bonutti, A., De Cesco, F., Di Gaspero, L., Schaerf, A.: Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Annals of Operations Research **194**(1), 59–70 (2012)
11. Burke, E.K., Marecek, J., Parkes, A.J., Rudová, H.: Decomposition, reformulation, and diving in university course timetabling. Computers & Operations Research **37**(3), 582–597 (2010)
12. Burke, E.K., Marecek, J., Parkes, A.J., Rudová, H.: A supernodal formulation of vertex colouring with applications in course timetabling. Annals of Operations Research **179**(1), 105–130 (2010)
13. Burke, E.K., Marecek, J., Parkes, A.J., Rudová, H.: A branch-and-cut procedure for the udine course timetabling problem. Annals of Operations Research **194**(1), 71–87 (2012)
14. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research **140**(2), 266–280 (2002)
15. Di Gaspero, L., McCollum, B., Schaerf, A.: The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical report, Queen's University, Belfast, United Kingdom (2007)
16. Di Gaspero, L., Schaerf, A.: Multi-neighbourhood local search with application to course timetabling. In: E.K. Burke, P.D. Causmaecker (eds.) Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), *Lecture Notes in Computer Science*, vol. 2740, pp. 262–275. Springer, Berlin Heidelberg (2003)
17. Di Gaspero, L., Schaerf, A.: Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modelling and Algorithms **5**(1), 65–89 (2006)
18. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electronic Notes in Theoretical Computer Science **89**(4), 543–560 (2003)
19. Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S.: Potassco User Guide. Institute for Informatics, University of Potsdam, second edition edn. (2015). URL http://potassco.sourceforge.net

20. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp series 3. In: F. Calimeri, G. Ianni, M. Truszczyński (eds.) Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15), *Lecture Notes in Artificial Intelligence*, vol. 9345, pp. 368–383. Springer-Verlag (2015)
21. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers (2012)
22. Gebser, M., Kaminski, R., Obermeier, P., Schaub, T.: Ricochet robots reloaded: A case-study in multi-shot ASP solving. In: T. Eiter, H. Strass, M. Truszczyński, S. Woltran (eds.) Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation: Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday, *Lecture Notes in Artificial Intelligence*, vol. 9060, pp. 17–32. Springer-Verlag (2015)
23. Gebser, M., Kaufmann, B., Otero, R., Romero, J., Schaub, T., Wanko, P.: Domain-specific heuristics in answer set programming. In: M. desJardins, M. Littman (eds.) Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI'13), pp. 350–356. AAAI Press (2013)
24. Geiger, M.J.: Applying the threshold accepting metaheuristic to curriculum based course timetabling - a contribution to the second international timetabling competition itc 2007. Annals of Operations Research **194**(1), 189–202 (2012)
25. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of the Fifth International Conference and Symposium on Logic Programming, pp. 1070–1080. MIT Press (1988)
26. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11), *Lecture Notes in Computer Science*, vol. 6683, pp. 507–523. Springer-Verlag (2011)
27. Lach, G., Lübbecke, M.E.: Curriculum based course timetabling: new solutions to udine benchmark instances. Annals of Operations Research **194**(1), 255–272 (2012)
28. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum **30**(1), 167–190 (2007)
29. Lü, Z., Hao, J.K.: Adaptive tabu search for course timetabling. European Journal of Operational Research **200**(1), 235–244 (2010)
30. Marler, R., Arora, J.: Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization **26**(6), 369–395 (2004)
31. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. CoRR **abs/0712.1097** (2007)
32. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Di Gaspero, L., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing **22**(1), 120–130 (2010)
33. Müller, T.: ITC2007 solver description: a hybrid approach. Annals of Operations Research **172**(1), 429–446 (2009)
34. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided maxsat resolution. In: C. Brodley, P. Stone (eds.) Proceedings of the Twenty-Eighth National Conference on Artificial Intelligence (AAAI'14), pp. 2717–2723. AAAI Press (2014)
35. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Mathematics and Artificial Intelligence **25**(3–4), 241–273 (1999)
36. Schaerf, A.: A survey of automated timetabling. Artificial Intelligence Review **13**(2), 87–127 (1999)
37. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: P. van Beek (ed.) Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05), *Lecture Notes in Computer Science*, vol. 3709, pp. 827–831. Springer-Verlag (2005)