Mutsunori Banbara | Takehide Soh
Naoyuki Tamura | Katsumi Inoue | Torsten Schaub

Answer set programming as a modeling language for course timetabling

Suggested citation referring to the original publication:
Theory and Practice of Logic Programming 13 (2013) 4-5, pp.783–798
DOI https://doi.org/10.1017/S1471068413000495
ISSN (print) 1471-0684
ISSN (online) 1475-3081

TLP **13** (4–5): Online Supplement, July 2013.

URL: http://dx.doi.org/10.1017/S1471068413000495

Answer set programming as a modeling language for course timetabling

MUTSUNORI BANBARA, TAKEHIDE SOH and NAOYUKI TAMURA

Kobe University, 1-1 Rokko-dai, Nada-ku, Kobe, Hyogo 657-8501, Japan
(*e-mail:* {banbara@,soh@lion.,tamura@}kobe-u.ac.jp)

KATSUMI INOUE

National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
(*e-mail:* inoue@nii.ac.jp)

TORSTEN SCHAUB

University of Potsdam, August-Bebel-Strasse 89, D-14482 Potsdam, Germany
(*e-mail:* torsten@cs.uni-potsdam.de)

submitted 10 April 2013; revised 23 June 2013; accepted 5 July 2013

Abstract

The course timetabling problem can be generally defined as the task of assigning a number of lectures to a limited set of timeslots and rooms, subject to a given set of hard and soft constraints. The modeling language for course timetabling is required to be expressive enough to specify a wide variety of soft constraints and objective functions. Furthermore, the resulting encoding is required to be extensible for capturing new constraints and for switching them between hard and soft, and to be flexible enough to deal with different formulations. In this paper, we propose to make effective use of ASP as a modeling language for course timetabling. We show that our ASP-based approach can naturally satisfy the above requirements, through an ASP encoding of the curriculum-based course timetabling problem proposed in the third track of the second international timetabling competition (ITC-2007). Our encoding is compact and human-readable, since each constraint is individually expressed by either one or two rules. Each hard constraint is expressed by using integrity constraints and aggregates of ASP. Each soft constraint S is expressed by rules in which the head is the form of `penalty(`S`,`V`,`C`)`, and a violation V and its penalty cost C are detected and calculated respectively in the body. We carried out experiments on four different benchmark sets with five different formulations. We succeeded either in improving the bounds or producing the same bounds for many combinations of problem instances and formulations, compared with the previous best known bounds.

KEYWORDS: answer set programming, educational timetabling, course timetabling

1 Introduction

Recent advances in Answer Set Programming (ASP) (Gelfond and Lifschitz 1988; Niemelä 1999; Baral 2003; Gebser *et al.* 2012) suggests a successful direction to extend logic programming to be more expressive and more efficient. ASP provides

a rich language and can be well suited for modeling combinatorial problems in computer science and artificial intelligence. Recent remarkable improvements in the efficiency of ASP solvers encourage researchers to solve many problems by ASP.

The general timetabling problem is known to be complex and difficult. Starting with (Gotlieb 1962), a great deal of research has been done on timetabling in several areas such as metaheuristics, integer programming, (constraint) logic programming, propositional satisfiability (SAT), and constraint programming (Schaerf 1999; Burke and Petrovic 2002; Lewis 2007). In recent years, timetabling has become an area of increasing interest in an international community involving both researchers and practitioners, such as the international series of PATAT conferences. The typical topics of this area include *educational timetabling*, *transport timetabling*, *employee timetabling*, *sports timetabling*, and so on. In this paper we consider an educational timetabling problem.

The educational timetabling problem can be 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. The hard constraints must be strictly satisfied. The soft constraints are not necessarily satisfied but the sum of violations should be desirably minimized. Usually the educational timetabling problems can be classified into three categories: *school timetabling*, *examination timetabling*, and *course timetabling*. The course timetabling problems can be further classified into two sub-categories: *curriculum-based course timetabling* and *post-enrolment course timetabling*.

The modeling language and problem modeling play a very important role in the real-world timetable generation (McCollum 2007). The latter is particularly challenging because different institutions have their own needs and policies, and problem *formulation* (a specific set of soft constraints) may change from institution to institution and from time to time. There have been therefore several proposals for problem modeling of the educational timetabling problems (Faber *et al.* 1998; Carter 2001; Burke and Petrovic 2002; Daskalaki and Birbas 2005; Qualizza and Serafini 2005; Schimmelpfeng and Helber 2007; Burke *et al.* 2010a; Burke *et al.* 2010b; Achá and Nieuwenhuis 2012; Burke *et al.* 2012; Lach and Lübbecke 2012). However, these works were mostly done in the area of integer programming, and there is very little literature on ASP.

In this paper, we propose to make effective use of ASP as a modeling language for course timetabling. The modeling language for course timetabling is required to be expressive enough to specify a wide variety of soft constraints and objective functions that reflect the real-world scenarios. Furthermore, the resulting encoding is required to be extensible for capturing new constraints and for switching them between hard and soft, and to be flexible enough to deal with different formulations. We show that our ASP-based approach can naturally satisfy the above requirements. Consequently, it enables a timetable keeper to rapidly specify problems and to experiment with different formulations at a purely declarative level. ASP solvers are then used for finding and enumerating solutions without the need of developing dedicated algorithms.

We present an ASP encoding of the curriculum-based course timetabling (CB-CTT) problem proposed in the third track of the second international timetabling competition (ITC-2007)(Gaspero *et al.* 2007; McCollum *et al.* 2010). Our encoding is compact and human-readable, since each constraint is individually expressed by either one or two rules. Each hard constraint is expressed by integrity constraints and aggregates of ASP. For the soft constraints, we use the predicate `penalty(`S`,`V`,`C`)` which is intended to express that a soft constraint S is violated by V and its penalty cost is C. Each soft constraint S is expressed by rules in which the head is the form of `penalty(`S`,`V`,`C`)`, and a violation V and its penalty cost C are detected and calculated respectively in the body.

To evaluate the efficiency of our proposed encoding, we carried out experiments on four different benchmark sets (36 instances in total) with 5 different formulations. For the tested 180 combinations, we succeeded either in improving the bounds or producing the same bounds for 70 combinations (39% in the total), compared with the previous best known bounds. More precisely, our encoding was able to improve the bounds for 34 combinations and to prove that 13 of them are optimal. It was also able to produce the same bounds for 36 combinations and to prove that 3 of them are newly optimal.

2 Curriculum-based course timetabling

2.1 Problem definition

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 violations should be desirably minimized. 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 (Bonutti *et al.* 2012).

- ***H*$_1$. Lectures**: All lectures of each course must be scheduled, and they must be assigned to distinct timeslots.
- ***H*$_2$. Conflicts**: Lectures of courses in the same curriculum or taught by the same teacher must be all scheduled in different timeslots.
- ***H*$_3$. RoomOccupancy**: Two lectures can not take place in the same room in the same timeslot.

- H_4. **Availability**: If the teacher of the course is not available to teach that course at a given timeslot, then no lecture of the course can be scheduled at that timeslot.
- 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.
- 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 in company with the weights associated with each of them. 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 (Gaspero and Schaerf 2003). UD2 is a formulation used in ITC-2007 (Gaspero *et al.* 2007). To capture more different

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	-

scenarios, UD3, UD4, and UD5 are proposed recently (Bonutti *et al.* 2012). 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.

2.2 Problem instance example

Figure 1 shows a tiny instance `toy.ectt` written in the '`.ectt`' format, a standard input format of the CB-CTT problem (Bonutti *et al.* 2012). Converting a '`.ectt`' instance to ASP facts is straightforward. Figure 2 shows an ASP representation of `toy.ectt`.

- The first nine facts 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 4 periods per day, where they start from 0.
- The fact `course(`C`,`T`,`N`,`MWD`,`M`,`DL`)` 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`)` expresses that a room R located in a building BLD has a seating capacity of CAP. The fact `curricula(`CUR`, `C`)` expresses that a curriculum CUR includes a course C.
- The fact `unavailability_constraint(`C`,`D`,`P`)`, 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`)`, which is used to specify S_8, expresses that a room R is not suitable for a course C.

As an output example, Figure 3 shows an optimal solution with zero cost of the tiny instance `toy.ectt` with the UD2 formulation. In this solution, all three

```
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.
```

Fig. 1. `toy.ectt`: Input example.

lectures of the course `SceCosC` are assigned to the room `rB` at the third period (`2`) on Wednesday (`2`), the first period (`0`) on Thursday (`3`), and the third period (`2`) on Friday (`4`).

3 Encoding of hard constraints

We present two different encodings called *Direct encoding* and *Linked encoding*. In our encodings, the hard constraints can be compactly expressed by using integrity constraints and aggregates of ASP. We use the syntax supported by the grounder gringo and the solver clasp (Gebser *et al.* 2007; Gebser *et al.* 2009).

Figure 4 shows common auxiliary rules shared by our two encodings. Given a problem instance, for each course `C`, teacher `T`, room `R`, and curriculum `Cu`, the first four rules generate `c(C)`, `t(T)`, `r(R)`, and `cu(Cu)` respectively. In the last two rules, `d(0..D-1)` and `ppd(0..P-1)` express that the days are integers in the range `0` to `D-1`, and the periods per day are integers in the range `0` to `P-1`.

Direct encoding. The most direct modeling would be using a quaternary predicate `assigned/4`. The predicate `assigned(`C`,`R`,`D`,`P`)` is intended to express that a

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

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

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

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

unavailability_constraint("TecCos",2,0).
unavailability_constraint("TecCos",2,1).
unavailability_constraint("TecCos",3,2).
unavailability_constraint("TecCos",3,3).
unavailability_constraint("ArcTec",4,0).
unavailability_constraint("ArcTec",4,1).
unavailability_constraint("ArcTec",4,2).
unavailability_constraint("ArcTec",4,3).

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

Fig. 2. ASP input example for `toy.ectt`.

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

Fig. 3. ASP output example for `toy.ectt` in UD2.

lecture of a course C is assigned to a room R at a period P on a day D. Figure 5 shows an ASP encoding of the hard constraints (H_1–H_4). It uses special constructs called *cardinality expressions* of the form $\ell\{a_1,\ldots,a_k\}u$ where each a_i is an atom and ℓ and u are non-negative integers denoting the lower bound and the upper bound of the cardinality expression. For H_1, the first rule, for every course `C` having `N` lectures, generates a solution candidate at first and then constrains that there are exactly `N` lectures such that `assigned(C,R,D,P)` holds. The second rule constrains that, for every course `C`, day `D`, and period `P`, there is at most one room `R` such that `assigned(C,R,D,P)` holds. For H_2, the third rule 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,R,D,P)` holds. The fourth rule 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,R,D,P)` holds. For H_3, the fifth rule 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 sixth rule constrains that, for

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

Fig. 4. Auxiliary rules.

```
% H1. Lectures
N { assigned(C,R,D,P) : r(R) : d(D) : ppd(P) } N :- course(C,_,N,_,_,_).
:- not { assigned(C,R,D,P) : r(R) } 1, c(C), d(D), ppd(P).

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

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

% H4. Availability
:- assigned(C,R,D,P), r(R), unavailability_constraint(C,D,P).
```

Fig. 5. Direct encoding.

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.

Linked encoding. It is obvious that we do not always have to take account of the room information to specify the hard constraints except H_3. Figure 6 shows another ASP encoding. The difference from the direct encoding is that we use a ternary predicate `assigned/3` in addition to `assigned/4`. The predicate `assigned(`C`,`D`,`P`)` is intended to express that a lecture of a course C is assigned to a period P on a day D. The hard constraints except H_3 are expressed by the first three and sixth rules that are slightly modified to adjust the predicate `assigned/3` by just deleting `r(R)` from the corresponding rules of the direct encoding. For H_3, the fourth rule first 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. That is, the predicate `assigned/3` is linked to `assigned/4` in this rule. The fifth rule is the same as one of the direct encoding. In the linked encoding, the constraints can be expressed more concisely by using different predicates for each, than the direct encoding. In addition, the following rule ‘`:- not { assigned(C,D,P) : c(C) } N, d(D), ppd(P), rooms(N).`’

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

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

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

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

Fig. 6. Linked encoding.

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. However, we use it as an additional rule, since it gives a performance improvement for some problem instances.

To evaluate the efficiency of our proposed encodings above, we carry out experiments on four different benchmark sets: `ITC-2007` (Gaspero *et al.* 2007) consisting of 21 instances denoted by `comp*`, `DDS-2008` (Bonutti *et al.* 2012) of 7 instances by `DDS*`, `Erlangen` of 4 instances by `erlangen*`, and `Test` (Gaspero and Schaerf 2003) of 4 instances by `test*`. These instances are based on real data from several European universities. 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. More detailed features are shown in Bonutti *et al.* (2012). All the instances are available from a web portal (`http://tabu.diegm.uniud.it/ctt/`) maintained by the organizers of ITC-2007.

We solve the instances as decision problems by taking only the hard constraints into account. We use the grounder gringo 3.0.4 and the solver clasp 2.1.1 (Gebser *et al.* 2007; Gebser *et al.* 2009) on Mac OS X with 1.8 GHz Intel Core i7 and 4 GB memory.

For each encoding, we were able to find a feasible solution in 180 seconds for every instance. Table 2 shows CPU time in seconds and the number of choices, conflicts, and restarts obtained by using the option `--stats` of clasp. We observe in Table 2 that the linked encoding is faster and can be more scalable to the number of courses than the direct encoding. The linked encoding is 3 times faster, 12, 9, and 95 times smaller on the number of choices, conflicts, and restarts respectively than the direct encoding on the average. For the tested 36 instances, the linked encoding solved 22 instances with 0 restarts, compared with 6 instances of the direct encoding. Moreover, it solved very large instances `erlangen*` with either 0 or 2 conflicts, compared with more than 4,000 conflicts of the direct encoding. From these observations, we decide to adopt the linked encoding as a basis for expressing the soft constraints.

4 Encoding of soft constraints

We present an ASP encoding of the soft constraints based on the linked encoding. We use the ternary predicate `penalty(`S_i`,`V`,`C`)` that is intended to express that a constraint S_i is violated by V and its penalty cost is C. Each constraint S_i is 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*.

Figure 7 shows an ASP encoding of the soft constraints (S_1–S_9). The constants denoted by `penalty_of_*` indicate the weights associated with each soft constraint defined in Table 1. Due to the page limitation, we give a detailed explanation of S_1–S_3 that constitute the basic formulation UD1. For S_1, the

Table 2. *Benchmark results of different ASP encodings (Hard Constraints Only)*

	Direct encoding				Linked encoding			
Instance	CPU	#Choices	#Conflicts	#Restarts	CPU	#Choices	#Conflicts	#Restarts
comp01	2.070	93644	74068	11	0.200	1902	892	0
comp02	3.400	155236	50303	21	1.510	23443	16115	2
comp03	3.500	163311	65586	19	0.300	3500	79	0
comp04	2.480	125015	48818	10	0.350	4478	11	0
comp05	0.480	11949	2715	2	0.350	1420	12	0
comp06	1.910	91625	16541	6	2.110	27304	16255	2
comp07	7.320	137697	48158	4	3.610	43477	32106	3
comp08	3.680	180398	65095	14	0.370	4701	116	0
comp09	4.500	178973	66930	32	0.340	4104	17	0
comp10	0.900	33042	6623	0	2.280	23770	16604	2
comp11	1.840	65586	50515	14	0.220	1247	74	0
comp12	0.530	14264	1763	0	0.880	6408	3602	0
comp13	7.140	166161	65699	20	1.980	24907	16918	2
comp14	1.370	47933	18426	1	1.330	24608	16150	1
comp15	3.500	163311	65586	19	0.320	3500	79	0
comp16	5.080	230339	53636	6	2.110	26798	16718	2
comp17	1.840	79955	17091	11	2.120	23043	16692	1
comp18	0.140	7070	728	0	0.180	1619	219	0
comp19	4.810	215313	84309	32	0.290	3586	129	0
comp20	4.920	94172	32718	2	2.270	27068	17188	4
comp21	2.390	52274	24951	1	1.850	25410	17438	2
DDS1	132.830	25681926	1073907	2574	3.880	37030	17145	6
DDS2	0.770	18384	2659	0	0.660	2468	0	0
DDS3	0.370	20791	6081	1	0.330	3784	448	0
DDS4	34.180	2665514	69925	61	15.400	2516522	32848	2
DDS5	21.300	839860	76747	126	1.960	11465	14	0
DDS6	0.600	26894	4921	0	2.000	24237	16109	2
DDS7	3.320	73722	48379	3	0.350	3166	22	0
erlangen2011_2	114.330	3151932	32460	1	30.960	140993	2	0
erlangen2012_1	24.850	1230689	4227	0	25.020	87210	0	0
erlangen2012_2	57.830	2776461	18465	1	28.890	116116	0	0
erlangen2013_1	41.280	3321458	18412	1	26.780	107708	2	0
test1	1.640	50949	33311	2	0.100	1637	0	0
test2	2.370	121361	66109	22	0.180	2148	118	0
test3	3.560	192608	112756	32	0.160	1931	12	0
test4	0.800	53703	21070	18	1.330	19719	16008	1
Average	13.995	1181487	68047	85.194	4.527	93956	7504	0.889

first rule, 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 `penalty_of_room_capacity`, if `N > Cap` and `assigned(C,R,D,P)` holds. For S_2, we use the second rule as an auxiliary rule. It 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 third rule, 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 `penalty_of_min_working_days`, if the number of days (`N`) in which a course `C` spread is less than `MWD`. For S_3, we use the fourth rule as an auxiliary rule. It 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 fifth rule, for every curriculum `Cu`, day `D`, and period `P`, generates a penalty atom with the constant cost `penalty_of_isolated_lectures`, 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`.

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

% S2. MinWorkingDays
working_day(C,D) :- assigned(C,D,P).
penalty("MinWorkingDays",course(C,MWD,N),(MWD-N)*penalty_of_min_working_days) :-
        course(C,_,_,MWD,_,_), N = [ working_day(C,_) ], N < MWD.

% S3. IsolatedLectures
scheduled_curricula(Cu,D,P) :- assigned(C,D,P), curricula(Cu,C).
penalty("IsolatedLectures",isolated_lectures(Cu,D,P),penalty_of_isolated_lectures) :-
        scheduled_curricula(Cu,D,P), not scheduled_curricula(Cu,D,P-1),
        not scheduled_curricula(Cu,D,P+1).

% S4. Windows
penalty("Windows",windows(Cu,C1,C2,D,P1,P2),(P2-P1-1)*penalty_of_windows) :-
        curricula(Cu,C1), curricula(Cu,C2), assigned(C1,D,P1), assigned(C2,D,P2),
        P1 + 1 < P2, not scheduled_curricula(Cu,D,P) : P = P1+1..P2-1.

% S5. RoomStability
using_room(C,R) :- assigned(C,R,D,P).
penalty("RoomStability",using_room(C,N),(N-1)*penalty_of_room_stability) :-
        c(C), N = [ using_room(C,_) ], N > 1.

% S6. StudentMinMaxLoad
penalty("StudentMinMaxLoad",student_min_max_load(Cu,D,N,many),(N-Max)*penalty_of_student_min_max_load) :-
        cu(Cu), d(D), N = { assigned(C,D,P) : curricula(Cu,C) : ppd(P) },
        min_max_daily_lectures(Min,Max), N > Max.
penalty("StudentMinMaxLoad",student_min_max_load(Cu,D,N,few),(Min-N)*penalty_of_student_min_max_load) :-
        cu(Cu), d(D), N = { assigned(C,D,P) : curricula(Cu,C) : ppd(P) },
        min_max_daily_lectures(Min,Max), 0 < N, N < Min.

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

% S8. RoomSuitability
penalty("RoomSuitability",assigned(C,R,D,P),penalty_of_room_suitability) :-
        assigned(C,R,D,P), room_constraint(C,R).

% S9. DoubleLectures
penalty("DoubleLectures",non_grouped_lecture(C,R,D,P),penalty_of_double_lectures) :-
        course(C,_,_,_,_,1), d(D), 2 [ assigned(C,D,_) ],
        assigned(C,R,D,P), not assigned(C,R,D,P-1), not assigned(C,R,D,P+1).
```

Fig. 7. Encoding of soft constraints.

```
#minimize [ penalty(_,_,P) = P ].
```

Fig. 8. Encoding of objective function.

5 Full encoding

The objective of the CB-CTT problem is to find a feasible solution of minimal penalty costs. The objective function of the problem is expressed by only one rule in Figure 8.

Full linked encoding consists of the rules of the auxiliaries (Fig. 4), the hard constraints (Fig. 6), the soft constraints (Fig. 7), and the objective function (Fig. 8). Now we go back to the requirements for the modeling language and problem encoding of course timetabling mentioned in Section 1, and see how they are satisfied in our approach. First, ASP is expressive enough to specify a wide variety of soft constraints and objective functions, since it can naturally express not only the classical constraints S_1–S_3 but also the relatively new constraints S_4–S_9. ASP also supports multi-criteria optimization. Second, our encoding is extensible enough for capturing new constraints, since the constraints can be compactly expressed by ASP,

and all we have to do for new constraints is adding rules. It is also extensible enough for switching constraints between hard and soft. For example, S_8 is defined as a hard constraint in UD4 as can be seen in Table 1. For the direction from soft to hard, in this case, we only have to delete the head from the twelfth rule in Figure 7 like ':- assigned(C,R,D,P), room_constraint(C,R).'. Another method is known to just assign a big number to the penalty cost. It is also easy 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 sixth rule in Figure 6. Finally, our encoding is flexible enough to deal with different formulations, since the constraints are expressed individually by separable rules, and we can easily select necessary rules depending on each formulation.

Perhaps the most relevant works are problem encodings in the area of integer programming (Burke *et al.* 2010a; Burke *et al.* 2010b; Burke *et al.* 2012; Lach and Lübbecke 2012). These encodings use the binary variables $x_{C,D,P}$ and/or $x_{C,R,D,P}$ that correspond to the predicate `assigned(`C`,`D`,`P`)` and/or `assigned(`C`,`R`,`D`,`P`)` respectively. SAT/MaxSAT encodings (Achá and Nieuwenhuis 2012) also use the same binary variables, but requires expensive SAT encodings of cardinality constraints. The main advantage of our approach is not only a compact and declarative representation but also human-readability with the help of the direct symbolic processing, gained by using ASP as a modeling language.

6 Comparison

To evaluate the efficiency of our proposed full encoding, we carry out experiments on the same benchmark sets as in Section 3. The differences from Section 3 are that we solve 36 instances as optimization problems with five different formulations UD1–UD5. We set a timeout of 3 hours for each except 24 hours for `erlangen*`. All times were measured on Mac OS X with 2.66 GHz Intel Xeon and 24 GB memory.

Table 3 shows the best upper bounds obtained by our encoding, compared with the best known bounds available on the web (`http://tabu.diegm.uniud.it/ctt/`, last accessed on 22 June 2013). The symbols '>' and '=' indicate that our encoding produced the improved and the same bounds respectively, compared to the previous best known bounds. If followed by a superscript '*', these symbols indicate that our encoding proved the optimality of the obtained bounds. The symbol '*n.a*' indicates that the result is not available on the web.

For the tested 180 combinations, we succeeded either in improving the bounds or producing the same bounds for 70 combinations (39% in the total), compared with the previous best known bounds. More precisely, our encoding was able to improve the bounds for 34 combinations and to prove that 13 of them are optimal. That is, we found and proved new optimal solutions for 13 combinations. It was also able to produce the same bounds for 36 combinations and to prove that 3 of them are newly optimal. Furthermore, it was able to produce upper bounds for very large instances `erlangen*` with every formulation, and 16 of them were unsolvable before.

Table 3. *Comparison results on the best known bounds*

	UD1			UD2			UD3			UD4			UD5		
Instance	Best known		ASP	Best known		ASP	Best known		ASP	Best known		ASP	Best known		ASP
comp01	4	=	4	5	=	5	8		10	6		9	11		45
comp02	12		40	24		125	22	>	12	32		107	168		714
comp03	38		111	66		196	29		147	362		474	173		523
comp04	18	=	18	35		36	2	$=^*$	2	15	$>^*$	13	80		215
comp05	219		522	290		947	324		1232	260		584	658		2753
comp06	14		27	27		155	10	$>^*$	8	24		39	130		747
comp07	3	=	3	6		79	0	=	0	12	$>^*$	3	77		910
comp08	20	>	19	37		39	4	$>^*$	2	17	>	15	77		212
comp09	54		139	96		264	10	$>^*$	8	41		122	164		428
comp10	2	=	2	4	=	4	2	$>^*$	0	12	$>^*$	3	92		633
comp11	0	=	0	0	=	0	0	=	0	0	=	0	0	=	0
comp12	239		606	300		1114	92		1281	111		479	595		2180
comp13	32		61	59		112	26		63	45		109	157		488
comp14	27	=	27	51		52	0	=	0	18	=	18	109		541
comp15	38		111	66		196	28		118	34		129	216		656
comp16	11	=	11	18		28	6	$>^*$	4	16	$>^*$	7	127		914
comp17	30		75	56		171	14	$>^*$	12	29		58	181		818
comp18	34		98	62		184	0	=	0	27		93	144		509
comp19	32		41	57		91	28		93	35		123	147		619
comp20	2	=	2	4		80	6	$>^*$	0	18		168	196		2045
comp21	43		119	74		232	20	>	6	42		121	178		651
DDS1	39	$>^*$	38	48		87	3272		5727	2593	>	2278	2445		5214
DDS2	0	=	0	0	=	0	120		391	76		199	68		303
DDS3	0	=	0	0	=	0	22	=	22	11		12	22	=	22
DDS4	16		17	17		26	153		4057	124		6793	269		19988
DDS5	0	=	0	0	=	0	54		390	163		454	98		616
DDS6	0	=	0	0	=	0	0	=	0	5	$>^*$	0	116		1148
DDS7	0	=	0	0	=	0	34		432	25		382	61		631
erlangen2011_2	*n.a*	>	3061	1171		7017	*n.a*	>	8122	*n.a*	>	3152	*n.a*	>	8010
erlangen2012_1	*n.a*	>	2782	943		5716	*n.a*	>	7544	*n.a*	>	2694	*n.a*	>	7585
erlangen2012_2	*n.a*	>	3332	1310		10638	*n.a*	>	9731	*n.a*	>	4624	*n.a*	>	9081
erlangen2013_1	*n.a*	>	2608	1092		5476	*n.a*	>	7289	*n.a*	>	3553	*n.a*	>	7253
test1	212		316	224		383	200		286	208		370	232		491
test2	8	=	8	16		31	0	=	0	4	$=^*$	4	20		185
test3	35		38	67		172	18	$=^*$	18	21		28	97		194
test4	27		72	73		232	12		20	33		92	166		453

7 Conclusion

In this paper, we showed that ASP is an ideal modeling language for course timetabling, as demonstrated by our proposed encoding of the curriculum-based course timetabling problem. In our experiments, we succeeded either in improving the bounds or producing the same bounds for many combinations of problem instances and formulations, compared with the previous best known bounds. All source code is available from `http://kaminari.istc.kobe-u.ac.jp/resource/ctt/cttasp-0.8.tgz`.

The course timetabling problem is known to be difficult, since it contains both hard constraints and several types of soft constraints. ASP is useful to handle those mixed types of constraints. That is why we can gain good performance in this problem. Future works include using the *weak constraints* of ASP-Core-2 that `gringo-4` supports for expressing soft constraints of the problem. Our ASP-based approach can be applied to a wide range of combinatorial optimization problems such as the other timetabling problems and the resource-constrained project scheduling problem(Schutt *et al.* 2011).

Finally, we discuss some more details of our experimental results on ASP solvers. We used the default search strategy of clasp for finding optimal solutions. However, we met the difficulty of decreasing the upper bounds sufficiently. Particularly in UD2 and UD5, a large number of bounds obtained by our encoding are far from the current best known bounds. This shows a limitation of our approach at present. To overcome this issue, there might be at least two approaches. One is finding the best configuration of clasp, since it offers several options which control search strategy. In further experiments, for some instances with UD2, we used the option `--opt-value` of clasp that initializes the objective function. We were then able to reproduce and re-prove the previously known optimal bounds of `comp16` and `comp20` in 6,630 and 409 seconds respectively. This result shows a possibility of further performance improvement, since these bounds were not obtained by the default setting. Another is building a solver portfolio including unsatisfiability-based ASP solvers. It would be promising because a portfolio-based solver claspfolio (Gebser *et al.* 2011) and an unsatisfiability-based solver unclasp (Andres *et al.* 2012) have been recently shown to be effective for ASP solving.

Acknowledgements

This work was partially funded by JSPS KAKENHI Grant Number 24300007 and DFG grant SCHA 550/9-1.

References

Achá, R. A. and Nieuwenhuis, R. 2012. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research (February 2012)*, 1–21.

Andres, B., Kaufmann, B., Matheis, O. and Schaub, T. 2012. Unsatisfiability-based optimization in clasp. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, A. Dovier and V. S. Costa, Eds. Leibniz International Proceedings in Informatics (LIPIcs), vol. 17, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 211–221.

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

Bonutti, A., Cesco, F. D., Gaspero, L. D. and Schaerf, A. 2012. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research 194,* 1, 59–70.

Burke, E. K., Marecek, J., Parkes, A. J. and Rudová, H. 2010a. Decomposition, reformulation, and diving in university course timetabling. *Computers & Operations Research 37,* 3, 582–597.

Burke, E. K., Marecek, J., Parkes, A. J. and Rudová, H. 2010b. A supernodal formulation of vertex colouring with applications in course timetabling. *Annals of Operations Research 179,* 1, 105–130.

Burke, E. K., Marecek, J., Parkes, A. J. and Rudová, H. 2012. A branch-and-cut procedure for the udine course timetabling problem. *Annals of Operations Research 194,* 1, 71–87.

Burke, E. K. and Petrovic, S. 2002. Recent research directions in automated timetabling. *European Journal of Operational Research 140,* 2, 266–280.

Carter, M. W. 2001. A comprehensive course timetabling and student scheduling system at the university of waterloo. In *Proceedings of the 3th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000)*, E. K. Burke and W. Erben, Eds. Lecture Notes in Computer Science, vol. 2079, Springer, 64–84.

Daskalaki, S. and Birbas, T. 2005. Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research 160,* 1, 106–120.

Faber, W., Leone, N. and Pfeifer, G. 1998. Representing school timetabling in a disjunctive logic programming language. In *Proceedings of the 13th Workshop on Logic Programming (WLP'98)*, U. Egly and H. Tompits, Eds. 43–52.

Gaspero, L. D., McCollum, B. and Schaerf, A. 2007. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical report, Queen's University, Belfast, United Kingdom. URL: `http://www.cs.qub.ac.uk/itc2007/curriculmcourse/report/curriculumtechreport.pdf`.

Gaspero, L. D. and Schaerf, A. 2003. Multi-neighbourhood local search with application to course timetabling. In *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002)*, E. K. Burke and P. D. Causmaecker, Eds. Lecture Notes in Computer Science, vol. 2740, Springer, Berlin Heidelberg, 262–275.

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

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M. T. and Ziller, S. 2011. A portfolio solver for answer set programming: Preliminary report. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, J. P. Delgrande and W. Faber, Eds. Lecture Notes in Computer Science, vol. 6645, Springer, 352–357.

Gebser, M., Kaufmann, B., Neumann, A. and Schaub, T. 2007. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, MIT Press, 386–392.

Gebser, M., Kaufmann, B. and Schaub, T. 2009. The conflict-driven answer set solver clasp: Progress report. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009)*, E. Erdem, F. Lin and T. Schaub, Eds. Lecture Notes in Computer Science, vol. 5753. Springer, 509–514.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, MIT Press, 1070–1080.

Gotlieb, C. C. 1962. The construction of class-teacher time-tables. In *Proceedings of IFIP Congress 62*, C. M. Popplewell, Ed. North-Holland, 73–77.

Lach, G. and Lübbecke, M. E. 2012. Curriculum based course timetabling: New solutions to udine benchmark instances. *Annals of Operations Research 194,* 1, 255–272.

Lewis, R. 2007. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum 30,* 1, 167–190.

McCollum, B. 2007. A perspective on bridging the gap between theory and practice in university timetabling. In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), Revised Selected Papers*, E. K. Burke and H. Rudová, Eds. Lecture Notes in Computer Science, vol. 3867, Springer, 3–23.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., Gaspero, L. D., Qu, R. and Burke, E. K. 2010. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing 22,* 1, 120–130.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 241–273.

Qualizza, A. and Serafini, P. 2005. A column generation scheme for faculty timetabling. In *Proceedings of the 5th international conference on the practice and theory of automated timetabling (PATAT 2004)*, E. K. Burke and M. A. Trick, Eds. Lecture Notes in Computer Science, vol. 3616, Springer, 161–173.

Schaerf, A. 1999. A survey of automated timetabling. *Artificial Intelligence Review 13,* 2, 87–127.

Schimmelpfeng, K. and Helber, S. 2007. Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum 29,* 4, 783–803.

Schutt, A., Feydy, T., Stuckey, P. J. and Wallace, M. G. 2011. Explaining the cumulative propagator. *Constraints 16,* 3, 250–282.