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

Postprint archived at the Institutional Repository of the Potsdam University in:
Postprints der Universität Potsdam
Mathematisch-Naturwissenschaftliche Reihe ; 594
ISSN 1866-8372
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-415469
DOI https://doi.org/10.25932/publishup-41546
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 $\text{penalty}(S,V,C)$, 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 $\text{penalty}(S, V, C)$ 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 $\text{penalty}(S, V, C)$, 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.
• **H4. 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.

• **S1. 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.

• **S2. 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.

• **S3. 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.

• **S4. 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.

• **S5. 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.

• **S6. 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.

• **S7. 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.

• **S8. 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.

• **S9. 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
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 H4, 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 S8, 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
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
ASP as a modeling language for course timetabling

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("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).

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 (H1–H4). 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 \( \ell \) 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 \( \text{assigned}(C,R,D,P) \) holds. The second rule constrains that, for every course \( C \), day \( D \), and period \( P \), there is at most one room \( R \) such that \( \text{assigned}(C,R,D,P) \) 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 \( \text{assigned}(C,R,D,P) \) holds. The fourth rule constrains that, for every curriculum \( C_u \), day \( D \), and period \( P \), there is at most one course \( C \) that belongs to \( C_u \) such that \( \text{assigned}(C,R,D,P) \) 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 \( \text{assigned}(C,R,D,P) \) holds. For \( H_4 \), the sixth rule 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 \texttt{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 \texttt{assigned/3} in addition to \texttt{assigned/4}. The predicate \texttt{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 \texttt{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 \texttt{assigned}(C,D,P) holds if \texttt{assigned}(C,D,P) holds. That is, the predicate \texttt{assigned/3} is linked to \texttt{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 { \texttt{assigned}(C,D,P) : c(C) } N, d(D), ppd(P), \texttt{rooms}(N).`
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 \( \text{assigned}(C,D,P) \) 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 \( \text{comp}^* \), DDS-2008 (Bonutti et al. 2012) of 7 instances by \( \text{DDS}^* \), Erlangen of 4 instances by \( \text{erlangen}^* \), and Test (Gaspero and Schaefer 2003) of 4 instances by \( \text{test}^* \). These instances are based on real data from several European universities. Among them the instances of Erlangen are very large. For example, the instance \( \text{erlangen2012}_2.\text{ectt} \) 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 \(--\text{stats} \) 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 \( \text{erlangen}^* \) 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 \( \text{penalty}(S_i, V, C) \) 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 \( \text{penalty}(S_i, V, C) \), 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 \( \text{penalty}(S_i, V, C) \) is generated. We refer to an instance of \( \text{penalty}/3 \) as a \textit{penalty atom}.

Figure 7 shows an ASP encoding of the soft constraints \( (S_1- S_9) \). The constants denoted by \( \text{penalty}\_\text{of}_{*} \) 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)

<table>
<thead>
<tr>
<th>Instance</th>
<th>Direct encoding</th>
<th>Linked encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>#Choices</td>
</tr>
<tr>
<td>comp01</td>
<td>2.070</td>
<td>93644</td>
</tr>
<tr>
<td>comp02</td>
<td>3.400</td>
<td>155236</td>
</tr>
<tr>
<td>comp03</td>
<td>3.500</td>
<td>163311</td>
</tr>
<tr>
<td>comp04</td>
<td>2.480</td>
<td>125015</td>
</tr>
<tr>
<td>comp05</td>
<td>0.480</td>
<td>11949</td>
</tr>
<tr>
<td>comp06</td>
<td>1.910</td>
<td>91625</td>
</tr>
<tr>
<td>comp07</td>
<td>7.320</td>
<td>137697</td>
</tr>
<tr>
<td>comp08</td>
<td>3.680</td>
<td>180398</td>
</tr>
<tr>
<td>comp09</td>
<td>4.500</td>
<td>178973</td>
</tr>
<tr>
<td>comp10</td>
<td>0.900</td>
<td>33042</td>
</tr>
<tr>
<td>comp11</td>
<td>1.840</td>
<td>65586</td>
</tr>
<tr>
<td>comp12</td>
<td>0.530</td>
<td>14264</td>
</tr>
<tr>
<td>comp13</td>
<td>7.140</td>
<td>166161</td>
</tr>
<tr>
<td>comp14</td>
<td>1.370</td>
<td>47933</td>
</tr>
<tr>
<td>comp15</td>
<td>3.500</td>
<td>163311</td>
</tr>
<tr>
<td>comp16</td>
<td>5.080</td>
<td>230339</td>
</tr>
<tr>
<td>comp17</td>
<td>1.840</td>
<td>79955</td>
</tr>
<tr>
<td>comp18</td>
<td>0.140</td>
<td>7070</td>
</tr>
<tr>
<td>comp19</td>
<td>4.810</td>
<td>215313</td>
</tr>
<tr>
<td>comp20</td>
<td>4.920</td>
<td>94172</td>
</tr>
<tr>
<td>DDS1</td>
<td>132.830</td>
<td>25681926</td>
</tr>
<tr>
<td>DDS2</td>
<td>0.770</td>
<td>18384</td>
</tr>
<tr>
<td>DDS3</td>
<td>0.370</td>
<td>20791</td>
</tr>
<tr>
<td>DDS4</td>
<td>34.180</td>
<td>2665514</td>
</tr>
<tr>
<td>DDS5</td>
<td>21.300</td>
<td>839860</td>
</tr>
<tr>
<td>DDS6</td>
<td>0.600</td>
<td>26894</td>
</tr>
<tr>
<td>DDS7</td>
<td>3.320</td>
<td>73722</td>
</tr>
<tr>
<td>erlangen2011_2</td>
<td>114.330</td>
<td>3151932</td>
</tr>
<tr>
<td>erlangen2012_1</td>
<td>24.850</td>
<td>1230689</td>
</tr>
<tr>
<td>erlangen2013_1</td>
<td>57.830</td>
<td>2776461</td>
</tr>
<tr>
<td>test1</td>
<td>1.640</td>
<td>50949</td>
</tr>
<tr>
<td>test2</td>
<td>2.370</td>
<td>121361</td>
</tr>
<tr>
<td>test3</td>
<td>3.560</td>
<td>192608</td>
</tr>
<tr>
<td>test4</td>
<td>0.800</td>
<td>53703</td>
</tr>
</tbody>
</table>

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 \( \text{Cap} \), generates a penalty atom with the cost of the production of \( N-\text{Cap} \) and \( \text{penalty\_of\_room\_capacity} \), if \( N > \text{Cap} \) and \( \text{assigned}(C,R,D,P) \) holds. For \( S_2 \), we use the second rule as an auxiliary rule. It generates an atom \( \text{working\_day}(C,D) \) for every course \( C \), day \( D \), and period \( P \) if \( \text{assigned}(C,D,P) \) holds. The atom \( \text{working\_day}(C,D) \) expresses that a course \( C \) is given on a day \( D \). The third rule, for every course \( C \) whose lectures must be spread into \( \text{MWD} \) days, generates a penalty atom with the cost of the production of \( \text{MWD}-N \) and \( \text{penalty\_of\_min\_working\_days} \), if the number of days (\( N \)) in which a course \( C \) spread is less than \( \text{MWD} \). For \( S_3 \), we use the fourth rule as an auxiliary rule. It generates an atom \( \text{scheduled\_curricula}(Cu,D,P) \) for every curriculum \( Cu \), course \( C \) that belongs to \( Cu \), day \( D \), and period \( P \) if \( \text{assigned}(C,D,P) \) holds. The atom \( \text{scheduled\_curricula}(Cu,D,P) \) 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 \( \text{penalty\_of\_isolated\_lectures} \), 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 \( D \).
ASP as a modeling language for course timetabling

% 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)\*penalty_of_min_working_days) :-
    course(C,\_,\_,\_,\_,\_), 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\_,\_), (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), 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,D,R1), assigned(C2,D,R2),
    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,\_,\_,\_,\_,\_), 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 \( \text{assigned}(C,D,P) \) and/or \( \text{assigned}(C,R,D,P) \) 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.
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/cttasp0.8.tgz.

7 Conclusion

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


