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

Mutsunori Banbara · Katsumi Inoue · Benjamin Kaufmann · Tenda Okimoto · 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. Furthermore, we extend the *teaspoon* system to multi-objective course timetabling and consider *minimal perturbation problems*.

Keywords Educational Timetabling · Course Timetabling · Answer Set Programming · Multi-objective Optimization · Minimal Perturbation Problems

1 Introduction

Educational timetabling (Burke and Petrovic, 2002; Lewis, 2007; Schaerf, 1999) 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

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

constraints. Hard constraints must be strictly satisfied. Soft constraints must not necessarily be satisfied but the overall number of violations should 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; (Bettinelli et al, 2015)), one of the most studied course timetabling problems.

The CB-CTT problems have been used in the third track of the second international timetabling competition (ITC-2007; (Di Gaspero et al, 2007; McCollum et al, 2010)). A web portal[1] for CB-CTT has been actively maintained by the ITC-2007 organizers (Bonutti et al, 2012). The web site provides 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 in Europe. The best known bounds on the web have been obtained by state-of-the-art CB-CTT solving techniques including the winner algorithm of ITC-2007: metaheuristics-based algorithms (Abdullah et al, 2012; Di Gaspero and Schaerf, 2003, 2006; Geiger, 2012; Lü and Hao, 2010), Integer Programming (Lach and Lübbecke, 2012), hybrid methods (Müller, 2009), SAT/MaxSAT (Achá and Nieuwenhuis, 2012), 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; (Baral, 2003; Gelfond and Lifschitz, 1988; Niemelä, 1999)) 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 (Erdem et al, 2016). 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 (Bonutti et al, 2012) 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*. 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 elaboration 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 (Gebser et al, 2015a). However, the question is whether the high-level approach of *teaspoon* matches the performance of dedicated systems. We empirically address this question by

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

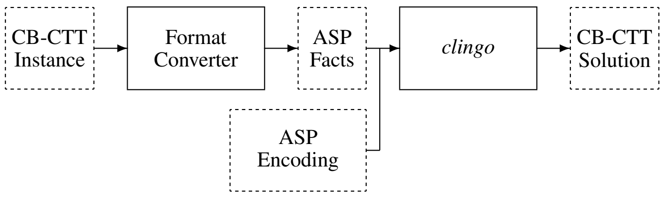

Fig. 1 Architecture of *teaspoon*.

contrasting the performance of *teaspoon* with the best known bounds on the CB-CTT web portal obtained by state-of-the-art CB-CTT solving techniques.

From the perspective of applying ASP to educational timetabling, an early work studied school timetabling with ASP (Faber et al, 1998). Recently, we showed in previous work (Banbara et al, 2013) that ASP's modeling language is well-suited for course timetabling by providing a compact encoding for CB-CTT solving. 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 main contributions of this paper are as follows.

1. We present a basic ASP encoding for solving CB-CTT problems, which is an enhancement of our previous encoding (Banbara et al, 2013). This enhancement provides the ability to use advanced ASP solving techniques such as core-guided optimization, domain heuristics, portfolios of prefabricated expert configurations, multi-criteria optimization based on lexicographic ordering, and multi-shot ASP solving (Gebser et al, 2015a,b).
2. We extend the basic encoding in view of enhancing the scalability and flexibility of solving (multi-criteria) CB-CTT problems. The extended *teaspoon* encodings have the following features:
   – A collection of optimized encodings for soft constraints
   – Easy composition of different formulations
   – Multi-criteria optimization based on lexicographic ordering
3. Our empirical analysis considers all 61 instances in 5 different formulations, which are publicly available from the CB-CTT portal ($61 \times 5 = 305$ combinations in a total) [2]. Overall, *teaspoon* managed to either improve or reproduce the best known bounds for 182 combinations (59.7% in the total). In detail, *teaspoon* provided 54 better bounds, 16 new optima, and 128 same bounds, 35 of which were proven optimal for the first time. Furthermore, *teaspoon* was able to produce upper bounds for very large instances in the category `erlangen` with every formulation, and 24 of them were unsolvable before.
4. We also extend the *teaspoon* system to finding Pareto optimal solutions of multi-objective course timetabling and consider *minimal perturbation problems* (Barták et al, 2004; Müller et al, 2005; Rudová et al, 2011; Phillips et al, 2016) by utilizing multi-shot ASP solving techniques (Gebser et al, 2015b).

All in all, the proposed declarative approach represents a significant contribution to the state-of-the-art for CB-CTT.

[2] As of July 20, 2017

The rest of the paper is structured as follows. Section 2 provides the problem description of CB-CTT. Although we give a brief introduction to ASP and its basic language constructs in Section 3, we refer the reader to the literature (Baral, 2003; Gebser et al, 2012) for a comprehensive treatment of ASP. Section 4 describes *teaspoon*'s fact format of CB-CTT instances and then presents a basic *teaspoon* encoding for solving CB-CTT problems. Section 5 presents a variety of features of extended *teaspoon* encodings for (multi-criteria) CB-CTT solving. Section 6 provides a detailed empirical analysis of *teaspoon* features and performance in contrast to the best known bounds obtained by state-of-the-art CB-CTT solving techniques. Section 7 presents an extension of the *teaspoon* system to minimal perturbation problems in course timetabling. Finally, a conclusion is given in Section 8.

2 Curriculum-based Course Timetabling

As mentioned, we focus on the curriculum-based course timetabling (CB-CTT) problems used in the ITC-2007 competition. The problem description of CB-CTT presented here is based on (Bonutti et al, 2012).

The CB-CTT instance consists mainly of *curricula*, *courses*, *rooms*, *days*, and *periods* per day. A curriculum is a set of courses that shares common students. We refer to a pair of day and period as *timeslot*. 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 hard and soft constraints. Hard constraints must be strictly satisfied. Soft constraints are not necessarily satisfied, but the sum of their violations should be minimal. A *feasible solution* of the problem is an assignment so that the hard constraints are satisfied. The objective of the problem is to find a feasible solution with minimal penalty. The CB-CTT problem has the following hard constraints.

$\boldsymbol{H_1}$. **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.

The CB-CTT problem has the following soft constraints.

$\boldsymbol{S_1}$. **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.
$\boldsymbol{S_2}$. **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.
$\boldsymbol{S_3}$. **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.
$\boldsymbol{S_4}$. **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

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	-

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.

A *formulation* is defined as a specific set of soft constraints together with the weights associated with each of them. The five formulations UD1–UD5 have been proposed so far. UD1 is the most basic formulation among them (Di Gaspero and Schaerf, 2003). UD2 is a well known formulation used in the ITC-2007 competition (Di Gaspero et al, 2007). UD3, UD4, and UD5 have been recently proposed to capture more different scenarios (Bonutti et al, 2012). These formulations focus on student load (UD3), double lectures (UD4), and travel cost (UD5), respectively. The weights of soft constraints in each formulation is shown in Table 1. The symbol ‘H’ stands for inclusion in a formulation as hard constraint. The symbol ‘-’ stands for exclusion from a formulation.

In this paper, we formulate the CB-CTT problem as a single-objective combinatorial optimization problem whose objective function is to minimize the weighted sum of penalty points in the same manner as ITC-2007, as well as a multi-criteria optimization problem based on lexicographic ordering. Furthermore, we consider a multi-objective course timetabling problem combining CB-CTT and Minimal Perturbation Problem.

3 Answer Set Programming

Answer Set Programming (ASP; (Baral, 2003; Gelfond and Lifschitz, 1988; Niemelä, 1999)) 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:

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

where $0 \leq m \leq n$ and each $\mathtt{a}_i$ is a propositional atom for $0 \leq i \leq n$. The connectives '`:-`' and '`,`' stand for 'if' and 'and', respectively. The connective '`not`' stands for *default negation*. Each rule is terminated by a period '`.`'. A *literal* is an atom `a` or `not a`. The intuitive meaning of the rule is that $\mathtt{a}_0$ must be true if $\mathtt{a}_1, \ldots, \mathtt{a}_m$ are true and if $\mathtt{a}_{m+1}, \ldots, \mathtt{a}_n$ are false. 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 (Gelfond and Lifschitz, 1988) for details.

We call a rule a *fact* if the *body* of the rule (right of '`:-`') is empty, and we often skip '`:-`' when writing facts. A rule is called an *integrity constraint* if the *head* of the rule (left of '`:-`') is empty.

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

A fact is unconditionally true, i.e., it belongs to every answer set. An integrity constraint is considered as a rule that filters solution candidates, meaning that the conjunction of literals in its body must not hold.

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* (Niemelä, 1999). The former are of the form `a:`$\mathtt{b}_1$`,...,`$\mathtt{b}_m$, the latter can be written as `s {`$\mathtt{c}_1$`,...,`$\mathtt{c}_n$`} t`, where `a` and $\mathtt{b}_i$ are possibly default-negated literals and each $\mathtt{c}_j$ is a conditional literal; `s` and `t` provide lower and upper bounds on the number of satisfied literals in the cardinality constraint. Note that either `s` or `t` can be omitted. That is, `s {`$\mathtt{c}_1$`,...,`$\mathtt{c}_n$`}` and `{`$\mathtt{c}_1$`,...,`$\mathtt{c}_n$`} t` represent at-least-`s` and at-most-`t` constraints respectively. 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 `N = {`$\mathtt{c}_1$`,...,`$\mathtt{c}_n$`}` that binds `N` to the number of satisfied conditional literals $\mathtt{c}_j$. Finally, objective functions minimizing the sum of weights $\mathtt{w}_j$ of conditional literals $\mathtt{c}_j$ are expressed as `#minimize {`$\mathtt{w}_1$`:`$\mathtt{c}_1$`,...,`$\mathtt{w}_n$`:`$\mathtt{c}_n$`}`. [4]

For solving a problem instance of a problem class in ASP, we encode the problem instance as a set of ASP facts and the problem class as a set of ASP rules. In turn, the facts

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

[4] Syntactically, each `wj` 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.

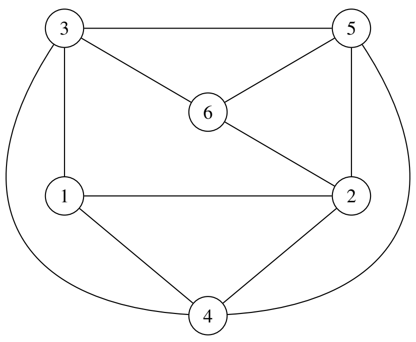

Fig. 2 Undirected graph having 6 nodes and 11 edges

```
node(1).    node(2).    node(3).   node(4).    node(5).    node(6).

edge(1,2).  edge(1,3).  edge(1,4). edge(2,4).  edge(2,5).  edge(2,6).
edge(3,4).  edge(3,5).  edge(3,6). edge(4,5).  edge(5,6).
```

Listing 1 ASP facts representing the graph of Figure. 2 (`graph.lp`)

```
1  col(r).   col(b).   col(g).

3  1 { color(X,C) : col(C) } 1 :- node(X).
4  :- edge(X,Y), color(X,C), color(Y,C).

6  #show color/2.
```

Listing 2 ASP rules for graph coloring (`color.lp`)

```
$ clingo graph.lp color.lp
clingo version 5.0.0
Reading from graph.lp ...
Solving...
Answer: 1
color(2,b) color(1,g) color(3,b) color(4,r) color(5,g) color(6,r)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Listing 3 Solving a graph coloring problem (`graph.lp` and `color.lp`)

are combined with the rules, and the result is subsequently solved by an off-the-shelf ASP system that returns an answer set representing a solution to the original problem.

As an example, let us consider a graph coloring problem. The problem consists in finding assignments of colors to nodes such that no two nodes connected by an edge have the same color. A problem instance is given by a graph as in Figure 2. It is represented as facts of predicates `node/1` and `edge/2` in Listing 1. The 3-colorability problem class is encoded in Listing 2. Line 1 provides the available colors as facts. Line 3 and 4 express the actual colorability problem. The predicate `color(X,C)` is used to express that a node `X` is colored

```
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 4 A toy instance of the `ectt` format

with `C`. The rule in Line 3 generates for each node `X` a set of candidate assignments subject to the condition that there is exactly one color `C` such that `color(X,C)` holds. In detail, the conditional literal `color(X,C):col(C)` in the cardinality constraint expands to the conjunction of `color(X,r)`, `color(X,b)`, and `color(X,g)`, since the facts `col(r)`, `col(b)`, and `col(g)` unconditionally hold. That is, line 3 expresses that each node `X` must be colored with exactly one color among red (`r`), blue (`b`), and green (`g`). The integrity constraint in Line 4 expresses that all connected nodes `X` and `Y` must not be colored with the same color `C`, since, as mentioned above, the conjunction of literals in its body must not hold. Line 6 is a directive, that is, a meta statement advising the ASP system to project answer sets onto instances of predicate `color/2`. An answer set computed by the ASP system *clingo* is shown in Listing 3; it represents a coloring of node `1` with green, `2` and `3` with blue, `4` with red, `5` with green, and `6` with red.

Modern ASP systems like *clingo* first translate user-defined logic programs (with variables) into equivalent ground (that is, variable-free) programs, and then compute the answer sets of the ground programs. Particularly, the former task is called *grounding*. The foundations and algorithms underlying the technology of *clingo* are described in detail in (Gebser et al, 2012).

4 The *teaspoon* Approach

We begin with describing *teaspoon*'s fact format of CB-CTT instances and then present a basic *teaspoon* encoding for solving the CB-CTT problems [5].

4.1 Fact Format

Listing 4 shows a toy instance of the `ectt` format, which is a standard input format of CB-CTT instances (Bonutti et al, 2012). The format has headers that represent basic entities,

[5] *teaspoon*: **T**im**E**tabling with **A**nswer **S**et **P**r**O**grammi**N**g

```
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 5 ASP facts representing the toy instance of Listing 4

```
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).
```

Listing 6 Solution (partial answer set) of the toy instance in UD2

followed by five blocks, `COURSES`, `ROOMS`, `CURRICULA`, `UNAVAILABILITY_CONSTRAINTS`, and `ROOM_CONSTRAINTS`.

ASP facts representing the toy instance are shown in Listing 5. There exists a one-to-one correspondence between ASP fact format and the `ectt` format except for the `CURRICULA` block. The facts in Line 1–2 correspond to the `ectt` headers and express that the 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, which start from 0. The fact `min_max_daily_lectures(2,3)` expresses the minimum and maximum numbers of daily lectures for each curriculum, and is used to specify S_6.

Each fact of predicate `course/6` in Line 4–5 corresponds to a line of the `COURSES` block. A fact `course(`C`,`T`,`N`,`MWD`,`M`,`DL`)` expresses that a course C taught by a teacher T consists of N lectures, which must be spread into MWD days. The number of students attending the course C is M. The course C requires double lectures if $DL = 1$. Each fact of predicate `room/3` in Line 7 corresponds to a line of the `ROOMS` block. A fact `room(`R`,`CAP`,`BLD`)` expresses that a room R in a building BLD has a seating capacity of CAP.

A fact `curricula(`CUR`, `C`)` in Line 9–10 expresses that a course C belongs to a curriculum CUR. Each fact of predicate `unavailability_constraint/3` in Line 12–15 corresponds to a line of the `UNAVAILABILITY_CONSTRAINTS` block, and is used to specify H_4. A fact `unavailability_constraint(`C`,`D`,`P`)` expresses that a course C is not available at a period P on a day D. Each fact of predicate `room_constraint/2` in Line 17 corresponds to a line of the `ROOM_CONSTRAINTS` block, and is used to specify S_8. A fact `room_constraint(`C`,`R`)` expresses that a room R is not suitable for a course C.

Listing 6 shows an optimal solution with zero penalty of the toy instance in the UD2 formulation. Each atom `assigned(`C`,`R`,`D`,`P`)` is intended to express that a lecture of a

```
 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 7 Encoding of hard constraints

course C is assigned to a room R at a period P on a day D. We can observe from Line 1 that the 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`)

4.2 First-Order Encoding

The *teaspoon* encoding of hard constraints (H_1–H_4) is shown in Listing 7. The expressive power of ASP's modelling language enables us to express each hard constraint individually by just one or two ASP rules. As mentioned, the atom `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 atom `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 an instance expressed in our 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)` expressing 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 set of candidate assignments subject to the condition that there are exactly `N` lectures such that `assigned(C,D,P)` holds.

For H_2, the rule in Line 9 enforces 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. In detail, if `t(T)`, `d(D)`, and `ppd(P)` hold, this integrity constraint tells us that the at-most-one constraint represented by '`{ assigned(C,D,P) : course(C,T,_,_,_,_) } 1`' must be true as well in order to prevent its body from being satisfied. In the similar way, the rule in Line 10 enforces 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, if `assigned(C,D,P)` holds, the rule in Line 13 generates a solution candidate subject to the condition that there is exactly one room `R` such that `assigned(C,R,D,P)`

```
% S1.RoomCapacity
penalty("RoomCapacity",assigned(C,R,D,P),(N-Cap)*weight_of_s1) :-
        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)*weight_of_s2) :-
        course(C,_,_,MWD,_,_), N = { working_day(C,D) }, N < MWD.

% S3.IsolatedLectures
scheduled_curricula(Cu,D,P) :- assigned(C,D,P), curricula(Cu,C).
penalty("IsolatedLectures",isolated_lectures(Cu,D,P),weight_of_s3) :-
        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,D,P1,P2),(P2-P1-1)*weight_of_s4) :-
        scheduled_curricula(Cu,D,P1), scheduled_curricula(Cu,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)*weight_of_s5) :-
        c(C), N = { using_room(C,R) }, N > 1.

% S6.StudentMinMaxLoad
penalty("StudentMinMaxLoad",student_min_max_load(Cu,D,N,many),(N-Max)*weight_of_s6) :-
        cu(Cu), d(D), N = { scheduled_curricula(Cu,D,P) },
        min_max_daily_lectures(Min,Max), N > Max.
penalty("StudentMinMaxLoad",student_min_max_load(Cu,D,N,few),(Min-N)*weight_of_s6) :-
        cu(Cu), d(D), N = { scheduled_curricula(Cu,D,P) },
        min_max_daily_lectures(Min,Max), 0 < N, N < Min.

% S7.TravelDistance
penalty("TravelDistance",instantaneous_move(Cu,C1,C2,D,P,P+1),weight_of_s7) :-
        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),weight_of_s8) :-
        assigned(C,R,D,P), room_constraint(C,R).

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

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

Listing 8 Encoding of soft constraints and objective function

holds. The rule in Line 14 enforces 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 enforces that a course `C` is not assigned at a period `P` on a day `D` if `unavailability_constraint(C,D,P)` holds, since the conjunction of literals in its body must not hold.

The rule in Line 20 expresses that for each timeslot (`D` and `P`) the number of lectures assigned must be less than or equal to the number of rooms (`N`). This rule is an implied constraint and can be omitted, but we keep it as an additional rule for performance improvement of some problem instances.

The *teaspoon* encoding of soft constraints (S_1–S_9) and an objective function is shown in Listing 8. We introduce a *penalty atom* `penalty(`S_i`,`V`,`C`)`, which is intended to express that a constraint S_i is violated by V and its penalty cost is C. The constants denoted by `weight_of_*` indicate the weights associated with each soft constraint defined in Table 1. Once again, each soft constraint S_i is compactly expressed by just one or two ASP rules in which the head is of the form `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, an atom `penalty(`S_i`,`V`,`C`)` is generated. Optimal solutions can be obtained by minimizing the number of penalty atoms in Line 49.

We explain S_1–S_3 that compose the basic UD1 formulation. For S_1, the rule in Line 2–3, 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 `(N-Cap)*weight_of_s1` if a lecture of course `C` is assigned to the room `R` whose seating capacity (`Cap`) is less than the number of attendees (`N`).

For S_2, the rule in Line 6 generates an auxiliary atom `working_day(C,D)` which expresses that a course `C` is given on a day `D`, if `assigned(C,D,P)` holds. 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 `(MWD-N)*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 auxiliary atom `scheduled_curricula(Cu,D,P)` which expresses that a curriculum `Cu` is scheduled at a period `P` on a day `D`, if `assigned(C,D,P)` holds. The rule in Line 12–14, for every curriculum `Cu`, day `D`, and period `P`, generates a penalty atom with the cost of `weight_of_s3`, if a curriculum `Cu` is scheduled at a period `P` on a day `D`, but not at both `P-1` and `P+1` within the same day `D`.

5 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. For scalability, we describe a collection of optimized *teaspoon* encodings for soft constraints in Section 5.1. For flexibility, we present significant extensions for easy composition of different formulations in Section 5.2 as well as for multi-criteria optimization based on lexicographic ordering in Section 5.3. And finally, we discuss the possibility of multi-shot ASP solving with *teaspoon* and illustrate a neighborhood search using (a part of) legacy timetables in Section 5.4.

5.1 Optimized encodings for soft constraints

The basic encoding in Section 4 precisely reflects the definition of CB-CTT constraints, but fails to scale to large instances in complex formulations like UD5 due to expensive grounding. To solve this practical issue, we present optimized encodings for the soft constraints S_2. MinWorkingDays, S_4. Windows, S_6. StudentMinMaxLoad, and S_7. TravelDistance.

For S_7, the rule in Line 35–37 of Listing 8 generates a penalty atom with the constant cost of `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 the same curriculum `Cu`, day `D`, and period `P`, and rooms `R1` and `R2` located in different buildings. This rule is very expensive when grounding due to its combinatorial blow-up caused by many variables. This issue can be improved by

```
 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 9 A collection of optimized encodings for S_2, S_4, S_6, and S_7

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 9. 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 of `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 9. The difference from the other two is that it utilizes cardinality constraints

```
% S8.RoomSuitability (soft constraint)
penalty("RoomSuitability",assigned(C,R,D,P),W,Prior) :-
        assigned(C,R,D,P), room_constraint(C,R), soft_constraint("RoomSuitability",W,Prior).

% S8.RoomSuitability (hard constraint)
:- assigned(C,R,D,P), room_constraint(C,R), hard_constraint("RoomSuitability").

% H4.Availability (soft constraint)
penalty("Availability",assigned(C,D,P),W,Prior) :-
        assigned(C,D,P), unavailability_constraint(C,D,P),
        soft_constraint("Availability",W,Prior).

% H4.Availability (hard constraint)
:- assigned(C,D,P), unavailability_constraint(C,D,P), hard_constraint("Availability").
```

Listing 10 Extended encoding of S_8 and H_4

```
soft_constraint("RoomCapacity",      1, 0). soft_constraint("MinWorkingDays",    1, 0).
soft_constraint("Windows",           1, 0). soft_constraint("StudentMinMaxLoad", 1, 0).
hard_constraint("RoomSuitability").         soft_constraint("DoubleLectures",    1, 0).
```

Listing 11 The UD4 formulation

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 9. 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 of `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 (Sinz, 2005), 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 is greater than or equal to N. The rule in Line 26–27 generates a penalty atom with the constant cost of `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`. The basic rules of S_6 is optimized in a similar way.

5.2 Easy composition of different formulations

Problem modeling is particularly challenging in the real-world course timetabling, since different institutions have their own needs and policies, and formulations (sets of constraints) may change from institution to institution and from time to time (McCollum, 2007). In this view, we present a design for easy composition of different formulations.

In order to easily activate or deactivate each soft constraint and switch it from soft to hard, we introduce new predicates `soft_constraint/3` and `hard_constraint/1`. The atom `soft_constraint`(S_i,W_i,L_i) is intended to express that S_i is a soft constraint to be

```
soft_constraint("RoomCapacity",      1, 0).  soft_constraint("MinWorkingDays",    1, 0).
soft_constraint("IsolatedLectures",  1, 0).  soft_constraint("Windows",           1, 0).
soft_constraint("RoomStability",     1, 0).  soft_constraint("StudentMinMaxLoad", 1, 0).
soft_constraint("TravelDistance",    1, 0).  soft_constraint("RoomSuitability",   1, 0).
soft_constraint("DoubleLectures",    1, 0).
```

Listing 12 Formulation consisting of all soft constraints (S_1–S_9) with the weights of all 1s

```
#minimize { P@L,C,S : penalty(S,C,P,L) }.
```

Listing 13 Objective function with priority levels for lexicographic optimization

```
soft_constraint("RoomCapacity",      1, 6).  soft_constraint("Windows",           2, 5).
soft_constraint("MinWorkingDays",    5, 4).  soft_constraint("TravelDistance",    2, 3).
soft_constraint("StudentMinMaxLoad", 2, 2).  soft_constraint("IsolatedLectures",  1, 1).
```

Listing 14 The UD5 formulation with different priority levels

activated. W_i and L_i respectively represent the weight and the priority level associated with S_i. The priority level L_i is used for lexicographic optimization explained later. 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*.

Listing 10 shows extended encodings of S_8.RoomSuitability and H_4.Availability with constraint atoms. For S_8, the rule in Line 2–3 is the same as before except that an instance of `soft_constraint/3` is added. Note that the penalty atom in its head is extended to `penalty/4` with a priority level L_i. The integrity constraint in Line 6 expresses S_8 as a hard constraint by dropping the penalty atom in the head in Line 2–3. On the other hand, it is also possible to switch constraints in the opposite direction, that is, from hard to soft. 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 7. An extended encoding of H_4 with constraint atoms is shown in Line 8–14 of Listing 10. The other constraints can be extended in a similar way.

The idea of constraint atoms allows for easy composition of different formulations, since any combination of constraints can be represented as a set of ASP 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 are shown in Listing 11. And also, we show exhaustive formulation consisting of all soft constraints (S_1–S_9) with the weights of all set to 1 in Listing 12[6]. These two examples represent single-objective weighted-sum formulations, since the priority levels are all 0.

5.3 Multi-criteria optimization based on lexicographic ordering

A well-known multi-criteria optimization strategy called lexicographic ordering (Marler and Arora, 2004) has been implemented in *clingo*. It enables us to optimize criteria in a lexicographic order based on their priorities. We here extend the *teaspoon* encoding for supporting such multi-criteria optimization.

[6] Surprisingly, the *teaspoon* system was able to find an optimal solution of `comp11` in this formulation.

This extension can be done by adding priority levels to the `#minimize` function, as can be seen in Listing 13. The variable `L` on the right-hand side of `@` stands for a priority level, where greater levels are more significant than smaller ones. Usually, solutions can be represented in the form of a utility vector $(p_1, p_2, \ldots, p_n)$, where each p_i is a value representing the penalty of a soft constraint.

Such lexicographic optimization is quite useful, since it enables a timetable keeper to experiment with different (pre-defined) priority levels of soft constraints. On the other hand, the optimality of multi-criteria optimization with lexicographic ordering does not always coincide with that of single-objective weighted-sum one. However, optimal solutions as well as better vectors obtained by lexicographic optimization can often correspond to feasible ones with smaller penalty in the original single-objective setting.

For illustration, Listing 14 shows an UD5 formulation with lexicographic optimization in which the priority levels of soft constraints are ordered as $S_1 > S_4 > S_2 > S_7 > S_6 > S_3$. In this case, the optimal solution of the ITC-2007 instance `comp13` is $(S_1, S_4, S_2, S_7, S_6, S_3) = (0, 0, 0, 0, 112, 35)$. This optimum corresponds to a smaller bound $112 + 35 = 147$ than the best known one in the CB-CTT portal obtained by single-objective weighted-sum optimization. A more detailed analysis of lexicographic optimization is shown in Section 6.3.

5.4 Discussion towards multi-shot ASP solving with *teaspoon*

Suppose that a (part of) legacy timetable is represented as a set of ASP facts of predicate `legacy/1`. ASP-based neighborhood search is implemented by only one rule:

```
#heuristic assigned(C,R,D,P) : legacy(assigned(C,R,D,P)). [1,true]
```

The special statement `#heuristic` is used to express various modifications to *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`.

Incremental SAT solving has recently been recognized as an important technique for many problems such as model checking and planning (Eén and Sörensson, 2003). From an ASP perspective, such multi-shot ASP solving is also available in *clingo* (Gebser et al, 2015b). It enables us to handle problem specifications which evolve during the reasoning process, because either 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 provides an incremental solving framework for finding optimal solutions as well as better bounds, while varying a set of constraints, switching them from hard to soft, varying the priority level of objectives, and searching neighborhoods by using a (part of) legacy timetables.

6 The *teaspoon* System

The *teaspoon* system accepts a standard input format, viz. `ectt` (Bonutti et al, 2012). 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.

6.1 Overview of Experiments

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` consists 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*`, `Udine` of 9 instances by `Udine*`, and the newest addition `UUMCAS_A131`. 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 [7] 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 or proven optimality for the same value. One configuration was automatically determined by *piclasp* 1.2.1, a configurator for *clingo* based on *smac* (Hutter et al, 2011). The parameter space was restricted to optimization related parameters and portfolio configurations. The `ITC-2007` instances served as training set[8] 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:

- ***BBn***: *Model-guided* branch-and-bound approach traditionally used for optimization in ASP. The idea is to iteratively produce models of descending cost until the optimal is found by establishing unsatisfiability of finding a model with lower cost. Parameter *n* controls how the cost is step-wise reduced, either strict lexicographically, hierarchically, exponentially increasing or exponentially decreasing.
- ***USCn***: *Core-guided* optimization techniques originated in MaxSAT (Biere et al, 2009). 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* (Andres et al, 2012), *pmres* (Narodytska and Bacchus, 2014), the combination of both with disjoint core preprocessing (Marques-Silva and Planes, 2007) and whether the constraints used to relax an unsatisfiable core are added as implications or equivalences. For $n > 8$, a technique called *stratification* (Ansótegui et al, 2013) 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 (Gebser et al, 2013).

[7] We used revision r10140 of the current development branch available at `https://potassco.org/`.

[8] 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	#Optima	#Unsolved	#Unique
VBS-ASP	12288.25	125	7	-
Best 14-way configuration	12370.44	125	7	-
Best 13-way configuration	12452.64	125	7	-
Best 12-way configuration	12596.48	125	7	-
Best 11-way configuration	12760.86	125	7	-
Best 10-way configuration	13004.02	125	7	-
Best 9-way configuration	13281.43	125	7	-
Best 8-way configuration	13589.65	125	7	-
Best 7-way configuration	13921.00	125	7	-
Best 6-way configuration	14510.06	125	7	-
Best 5-way configuration	15171.03	125	7	-
Best 4-way configuration	16354.07	122	7	-
Best 3-way configuration	17732.37	122	7	-
Best 2-way configuration	19606.93	122	7	-
USC11-JP	23321.47	122	8	20
USC11	23866.24	119	7	6
BB0-HEU3-RST	23960.18	77	7	23
USC13	24206.02	116	7	3
USC15	24250.30	117	7	2
USC13-CR	24324.16	116	23	5
USC15-CR	24343.91	118	22	4
USC11-CR	24377.86	116	23	3
USC13-HEU3-RST-HD (*LRND*)	24602.89	115	7	9
BB2-TR	24896.40	79	8	28
USC3-JP	25179.12	125	127	6
BB0	25281.20	73	7	14
USC1	25810.42	118	134	2
BB2	26662.98	78	7	6
Dom5	27310.77	76	160	7

– ***LRND***: Refers to the configuration automatically learned by *piclasp*. For space reasons, the configuration is referred 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 (Gebser et al, 2015a).

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*.

6.2 Evaluation of Basic Encoding

At first, we analyze the difference between the configurations using the basic encoding. 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 configurations are ordered by the mean rank that was calculated as suggested in

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

the ITC-2007 [9]. 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 solutions for each configuration.

The highest-ranked single (viz. 1-way) configuration was *USC11-JP*, though *USC3-JP* obtained the highest number of optimal solutions among the single configurations. Overall, *core-guided* strategies with *stratification* (*USCn with* $n > 8$) 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 (*USC11-JP*) and the best virtual configuration (*VBS-ASP*) 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 that row. The last column shows how many times the configuration in that row was in a best k-way configuration. The rows are ordered in descending order by the times the configuration in the row was included in a best k-way configuration. The only single configuration that is not contained in any k-way configuration is *USC1*. 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

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

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.

The time in seconds of finding optimal solutions for each combination of instance and formulation with *VBS-ASP* 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. With increasing formulation number, the number of optimal solutions decreases and, except for UD4, the average time increases.

We can observe from Table 2 that seven instances remained unsolved via the basic encoding for all configurations. Among them, the six `erlangen` instances in the UD5 formulation could not be grounded within a day and solving was therefore not possible. The new instances `UUMCAS_A131` in UD5 exceeded the memory limit of 20GB while solving.

6.3 Evaluation of Optimized and Extended Encodings

We evaluate the optimized encoding from Section 5.1 and the extended encoding using lexicographic optimization in Section 5.3. We utilize all 61 instances in the UD5 formulation as benchmark set. This is because UD5 is the hardest formulation and includes revised soft constraints: S_2, S_4, S_6 and S_7. To showcase the usefulness of the best k-way configurations, we use the *clingo*'s parallel search via multi-threading (Gebser et al, 2015a) and configure 8 threads using the best 8-way configuration. Since 8 cores are available, every thread will be able to run the instance simultaneously. Grounding and solving respectively were limited to 3 hours and 20GB memory usage[10].

Table 5 compares the basic encoding with the optimized encoding. The columns display in order the mean rank to compare the quality of the solutions, the average grounding time in seconds (denoted by T), the number of grounding timeouts (denoted by *#TO*), the number of optimal solutions, and the number of unsolved instances.

The optimized encoding is able to solve all 61 instances. In contrast, the basic encoding cannot find any feasible solutions for 11 instances due to the grounding timeouts for 6 and the solving timeouts for 5. The optimized encoding drastically reduces the grounding time compared to the basic encoding. While the number of optima obtained is identical, the lower mean rank shows that the optimized encoding improved upon the objective values obtained by the basic encoding.

[10] As mentioned, modern ASP systems like *clingo* first translate logic programs (with variables) into equivalent ground (that is, variable-free) programs, and then compute the answer sets of the ground programs. The former is called *grounding*, and the latter is called *solving*.

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

Table 5 Comparison between different encodings for UD5

Encoding	Mean rank	Grounding		#Optima	#Unsolved
		T (sec.)	*#TO*		
optimized	**1.43**	**21**	**0**	**9**	**0**
basic	1.57	1121	6	**9**	11

Table 6 The benchmark results of lexicographic optimization

Instance	Time (sec.)	The best utility vector						The sum of	The best of basic
		(S_1,	S_4,	S_2,	S_7,	S_6,	S_3)	utility vector	and optimized
`comp01`	10800.00	(4,	18,	55,	48,	40,	16)	181	**19**
`comp02`	10800.00	(0,	0,	10,	34,	138,	49)	**231**	338
`comp03`	**70.71**	(0,	0,	15,	0,	132,	57)	**204**	238
`comp04`	**7.16**	(0,	0,	0,	0,	38,	18)	56	**49**
`comp05`	10800.00	(0,	48,	145,	454,	534,	197)	1378	**1081**
`comp06`	10800.00	(0,	2,	10,	0,	52,	24)	**88**	253
`comp07`	10800.00	(0,	0,	0,	0,	190,	66)	**256**	603
`comp08`	**16.27**	(0,	0,	0,	0,	44,	21)	65	**55**
`comp09`	**529.40**	(0,	0,	20,	0,	122,	54)	**196**	215
`comp10`	10800.00	(0,	0,	0,	0,	54,	19)	**73**	285
`comp11`	**0.85**	(0,	0,	0,	0,	0,	0)	0	0
`comp12`	10800.00	(0,	0,	80,	838,	674,	304)	1896	**1135**
`comp13`	**4824.58**	(0,	0,	0,	0,	112,	35)	**147**	276
`comp14`	**1280.85**	(0,	0,	0,	0,	56,	28)	84	**67**
`comp15`	**3027.78**	(0,	0,	15,	48,	134,	57)	**254**	379
`comp16`	10800.00	(0,	0,	0,	120,	228,	90)	**438**	784
`comp17`	10800.00	(0,	0,	0,	80,	190,	82)	**352**	391
`comp18`	10800.00	(0,	0,	5,	0,	228,	109)	342	**228**
`comp19`	10800.00	(0,	0,	10,	42,	216,	79)	347	**283**
`comp20`	10800.00	(0,	0,	0,	420,	208,	76)	**704**	1098
`comp21`	**1342.53**	(0,	0,	10,	0,	110,	46)	**166**	215
`DDS1`	10800.00	(0,	1640,	135,	1618,	976,	293)	**4662**	4912
`DDS2`	10800.00	(0,	26,	0,	24,	96,	4)	**150**	165
`DDS3`	10800.00	(0,	0,	0,	0,	22,	0)	22	22
`DDS4`	10800.00	(15,	2362,	25,	1030,	672,	673)	4777	**2384**
`DDS5`	**66.13**	(0,	0,	0,	0,	88,	2)	90	**76**
`DDS6`	10800.00	(0,	0,	0,	258,	178,	73)	**509**	864
`DDS7`	10800.00	(0,	0,	0,	0,	64,	2)	**66**	218
`EA01`	10800.00	(0,	0,	45,	62,	180,	26)	**313**	645
`EA02`	10800.00	(0,	0,	5,	26,	132,	3)	**166**	487
`EA03`	10800.00	(0,	0,	10,	128,	470,	53)	**661**	1750
`EA04`	10800.00	(0,	0,	0,	0,	18,	0)	18	18
`EA05`	10800.00	(0,	0,	0,	0,	14,	0)	14	14
`EA06`	10800.00	(0,	0,	10,	64,	182,	7)	**263**	479
`EA07`	10800.00	(0,	0,	0,	0,	490,	21)	**511**	1681
`EA08`	10800.00	(0,	0,	0,	0,	40,	0)	40	40
`EA09`	10800.00	(0,	0,	0,	0,	46,	3)	49	48
`EA10`	10800.00	(0,	0,	0,	0,	228,	17)	**245**	298
`EA11`	10800.00	(0,	0,	0,	0,	40,	0)	**40**	73
`EA12`	10800.00	(0,	0,	0,	0,	26,	2)	**28**	40
`erlangen2011_2`	10800.00	(0,	0,	396,	165,	8052,	3740)	**12353**	17034
`erlangen2012_1`	10800.00	(0,	0,	6042,	70,	14938,	10413)	31463	**28236**
`erlangen2012_2`	10800.00	(0,	0,	11066,	75,	14792,	12029)	37962	**37103**
`erlangen2013_1`	10800.00	(0,	0,	6176,	75,	13724,	9472)	29447	**28997**
`erlangen2013_2`	10800.00	(0,	0,	7320,	80,	13664,	10531)	31595	**30533**
`erlangen2014_1`	10800.00	(0,	0,	7374,	95,	12158,	9356)	28983	**28655**
`test1`	10800.00	(218,	78,	140,	142,	84,	101)	763	**53**2
`test2`	**245.86**	(0,	0,	0,	0,	16,	8)	24	**20**
`test3`	**102.28**	(0,	0,	40,	0,	78,	39)	157	**68**
`test4`	10800.00	(0,	0,	35,	0,	164,	79)	**278**	401
`toy`	**0.18**	(0,	0,	0,	0,	0,	0)	0	0
`Udine1`	10800.00	(0,	0,	0,	0,	220,	14)	**234**	420
`Udine2`	10800.00	(0,	0,	0,	0,	104,	27)	**131**	264
`Udine3`	**562.73**	(0,	0,	0,	0,	34,	3)	37	37
`Udine4`	**5.43**	(0,	0,	2,	0,	88,	36)	126	**106**
`Udine5`	10800.00	(0,	0,	0,	0,	70,	8)	**78**	99
`Udine6`	10800.00	(0,	0,	0,	0,	36,	0)	**36**	41
`Udine7`	10800.00	(0,	0,	0,	0,	64,	0)	**64**	72
`Udine8`	10800.00	(0,	0,	0,	0,	132,	38)	**170**	172
`Udine9`	10800.00	(0,	0,	0,	15,	38,	3)	**56**	86
`UUMCAS_A131`	10800.00	(0,	7844,	0,	5108,	5414,	1333)	**19699**	19930

Next, we employ multi-criteria optimization based on lexicographic ordering of the soft constraints as described in Section 5.3. In our experiment, the soft constraints of the UD5 formulation are ordered as seen in Listing 14, viz. $S_1 > S_4 > S_2 > S_7 > S_6 > S_3$. We use the optimized version of the soft constraints S_2, S_4, S_6 and S_7.

Table 6 shows the benchmark results of lexicographic optimization. The columns display in order the solving time in seconds, the best utility vector of objective values which are ordered by priority levels, the sum of the vector constituting the single-objective point of view, and the best bounds obtained from the basic and optimized encodings with the single-objective weighted-sum optimization. The solving times of instances for which we were able to find the optimal vectors are highlighted in bold. The better bounds of the last two columns (viz. the single-objective case) are also highlighted.

The results show that *teaspoon* is capable of efficiently solving the multi-criteria optimization problem. It manages to find 15 optimal vectors. Furthermore, it obtains higher quality single-objective solutions for 33 instances. Note that the optimality of lexicographic optimization does not always entail that of the single-objective case, as can be seen for the instance `comp04`.

6.4 Comparison with other approaches

We compare the performance of *teaspoon* with other approaches. Table 7 [11] contrasts the best results of *teaspoon* including optimized encoding and lexicographic optimization, with the best known ones on the CB-CTT web portal [12] The symbols '$>$' and '$=$' indicate that *teaspoon* produced 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.

The basic encoding succeeded either in improving the bounds or producing the same bounds for 172 combinations (56.4% in the total), compared with the previous best known bounds. More precisely, the basic encoding was able to improve the bounds for 42 combinations and to prove that 15 of them are optimal. That is, we found and proved new optimal solutions for 15 combinations. It was also able to produce the same bounds for 130 combinations and to prove for the first time that 35 of them are optimal.

The optimized encoding produced one more new optimal solution, provided 7 better bounds, and produced the same bounds for 4 more instances. Furthermore, *teaspoon* was able to produce upper bounds for very large instances in the category `erlangen` with every formulation, and 24 of them were unsolvable before. Additionally, *teaspoon* was able to improve the bounds of 4 instances and achieve the same bound for one instance via lexicographic optimization.

Overall, *teaspoon* managed to either improve or reproduce the best known bounds for 182 combinations (59.7% in the total). In detail, *teaspoon* provided 54 better bounds, 16 new optima, and 128 same bounds, 35 of which were proven optimal for the first time.

Finally, we briefly compare the new results with our previous work (Banbara et al, 2013). The 185 benchmark combinations in (Banbara et al, 2013) were a subset of ones in Table 7,

[11] Among the six instances of `erlangen`, `erlangen2011_2`, `erlangen2012_1`, `erlangen2012_2`, and `erlangen2013_1` have been changed since 2014 due to the crush of the CB-CTT portal. We used new version of these instances in Table 7. Note that our previous works (Banbara et al, 2013, 2016) used old ones.

[12] `http://tabu.diegm.uniud.it/ctt/` as of July 20 2017

Table 7 Comparison of *teaspoon* with other approaches

Instance	UD1			UD2			UD3			UD4			UD5		
	Best known		*teaspoon*	Best known		*teaspoon*	Best known		*teaspoon*	Best known		*teaspoon*	Best known		*teaspoon*
`comp01`	4	=	4	5	=	5	8	=	8	6		9	11		19
`comp02`	12	=*	12	24	=*	24	12	=*	12	26		55	130		231
`comp03`	38		53	64		109	25		47	362		405	142		204
`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		88
`comp07`	3	=	3	6	=	6	0	=	0	3	=*	3	42		256
`comp08`	19	=	19	37	=	37	2	=*	2	15	=*	15	62	>*	55
`comp09`	54		63	96		169	8	=*	8	38		50	150		196
`comp10`	2	=	2	4	=	4	0	=	0	3	=*	3	72		73
`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	>	147
`comp14`	27	=*	27	51	=	51	0	=	0	16	>*	14	95	>*	67
`comp15`	38		53	62		109	16		22	30		68	176		254
`comp16`	11	=*	11	18	=	18	4	=*	4	7	=*	7	96		438
`comp17`	30	=*	30	56	=	56	12	=*	12	26	>*	21	155		352
`comp18`	34		48	61		81	0	=	0	27		46	137		228
`comp19`	32	>*	29	57	=	57	24		32	32		82	125		283
`comp20`	2	=	2	4	=	4	0	=	0	9	>*	3	124		704
`comp21`	43		94	74		124	6	=	6	36		76	151		166
`DDS1`	38	=*	38	48	=	48	2393		6036	2278	=	2278	1831		4662
`DDS2`	0	=	0	0	=	0	120		379	76		139	64		150
`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		509
`DDS7`	0	=	0	0	=	0	30		408	21		506	52		66
`EA01`	55	=*	55	65	=*	65	102		110	67		88	196		313
`EA02`	0	=	0	0	=	0	96		263	41		262	128		166
`EA03`	1	=	1	2	=	2	50		234	6936	>	816	90		661
`EA04`	0	=	0	0	=	0	18		21	9		695	18	=	18
`EA05`	0	=	0	0	=	0	14	=	14	7		8	14	=	14
`EA06`	5	=*	5	5	=*	5	42		156	27		336	99		263
`EA07`	0	=	0	0	=	0	206		1822	3884	>	1122	205		511
`EA08`	0	=	0	0	=	0	40		48	20		82	40	=	40
`EA09`	2	=*	2	4	=*	4	40	=	40	22		27	48	=	48
`EA10`	0	=	0	0	=	0	4		141	19		573	93		245
`EA11`	0	=	0	0	=	0	36		52	19		22	45	>	40
`EA12`	2	=*	2	4	=*	4	22		38	12		24	27		28
`erlangen2011_2`	*n.a*	>	3909	4670		11167	*n.a*	>	12790	*n.a*	>	6097	*n.a*	>	12353
`erlangen2012_1`	*n.a*	>	7207	7862		12563	*n.a*	>	18875	*n.a*	>	14212	*n.a*	>	28236
`erlangen2012_2`	*n.a*	>	12140	8813		23817	*n.a*	>	29169	*n.a*	>	18741	*n.a*	>	37103
`erlangen2013_1`	*n.a*	>	9415	7359		17730	*n.a*	>	20192	*n.a*	>	8201	*n.a*	>	28997
`erlangen2013_2`	*n.a*	>	9901	8150		19839	*n.a*	>	23285	*n.a*	>	12682	*n.a*	>	30533
`erlangen2014_1`	*n.a*	>	9205	5981		18395	*n.a*	>	20286	*n.a*	>	8048	*n.a*	>	28655
`test1`	212		328	224		404	200		299	208		413	232		532
`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		278
`toy`	0	=	0	0	=	0	0	=	0	0	=	0	0	=	0
`Udine1`	0	=	0	0	=	0	128		426	64		427	138		234
`Udine2`	4	=	4	8	=*	8	34		322	30		320	81		131
`Udine3`	0	=	0	0	=	0	24		88	19		67	54	>*	37
`Udine4`	35	=	35	64	=	64	24	=*	24	31	=*	31	108	>*	106
`Udine5`	0	=	0	0	=	0	44		338	23		145	47		78
`Udine6`	0	=	0	0	=	0	36		76	18		50	38	>	36
`Udine7`	0	=	0	0	=	0	64		94	32		62	64	=	64
`Udine8`	16	=*	16	31	>*	29	42		297	31		149	88		170
`Udine9`	18	=	18	21	=*	21	28		62	23		91	70	>	56
`UUMCAS_A131`	*n.a*	>	776	708	>	274	*n.a*	>	28088	*n.a*	>	10955	*n.a*	>	19699

```
penalty("ManhattanDistance",assigned(C,R,D,P),W,Prior) :-
        legacy(assigned(C,R,D,P)),
        not assigned(C,R,D,P),
        change_constraint("ManhattanDistance",W,Prior).
```

Listing 15 ASP Encoding of the Manhattan distance

comprised of the 5 formulations for the categories comp, DDS, test and erlangen without erlangen2013_2 and erlangen2014_1. Note that we ignore 20 combinations of all erlangen instances for comparison, since these instances are old and different from ones used in this paper. The *teaspoon* system was able to obtain the same or better bounds for 153 combinations (92.7% in the total). In detail, *teaspoon* improves bounds for 94 combinations and proves optimality for 31 of them. For 59 combinations, the same bounds were produced and 6 of them were confirmed to be optimal.

7 Multi-objective Course Timetabling considering Minimal Perturbation Problems

The Multi-Objective Discrete Optimization Problem (MODOP; (Ehrgott, 2005)) is a problem involving multiple objective functions that should be considered separately and optimized simultaneously. It is therefore well suited for modeling many real world applications involving multiple criteria, such as decision making, scheduling, automated planning, etc. The goal of this problem is finding the *Pareto front* (viz. the set of Pareto optimal solutions) defined by Pareto optimality. Usually, a Pareto optimal solution is expressed in the form of utility vector $(o_1, o_2, \ldots, o_n)$ where each o_i is a value representing the optimization quality of an objective. From a practical point of view, finding a promising subset of Pareto optimal solutions is also important, since finding the Pareto front is known to be difficult.

The Minimal Perturbation Problem (MPP; (Sakkout and Wallace, 2000; Barták et al, 2004; Zivan et al, 2011)) can be generally defined as a problem aiming at solutions featuring minimal perturbation with respect to an initial solution. For a given initial solution of the original problem, a new problem, and a distance function that is used to compare a solutions of the new problem with the initial solution, the goal of MPP is to find a solution of the new problem with a minimal distance to the initial solution of the original problem. There exists some work on MPP in course timetabling (Rudová et al, 2011; Müller et al, 2005; Phillips et al, 2016). In this case, the goal is to find a feasible solution of a new timetabling problem with minimal change usually defined by *Manhattan distance* to the legacy solution.

In this section, we extend the *teaspoon* system to solving multi-objective course timetabling problems combining CB-CTT and MPP with two criteria *optimality* and *stability*. The first criterion, optimality, is to minimize the penalty of soft constraints for CB-CTT solving. The second criterion, stability, is to minimize the change from a legacy solution for MPP solving. For a given legacy solution of the original CB-CTT problem, a new CB-CTT problem, and a distance function, the goal is to find Pareto optimal solutions, that is, trade-off solutions of the new problem with minimal penalty as well as with minimal change from the legacy solution. We adopt the Manhattan distance as a distance function, which is widely used in MPP research. Pareto optimal solutions of this problem are in the form of a utility vector (o_1, o_2) in which o_1 is a value for optimality representing the optimization quality, and o_2 is a value for stability representing the stabilization quality.

```
soft_constraint("RoomCapacity",      1, p1).
soft_constraint("MinWorkingDays",    5, p1).
soft_constraint("IsolatedLectures", 2, p1).
soft_constraint("RoomStability",     1, p1).

change_constraint("ManhattanDistance", 1, p2).
```

Listing 16 The UD2 formulation with the Manhattan distance

For optimality, we can use *teaspoon* encodings presented in Section 4 and Section 5. For stability, we explain *teaspoon*'s fact format of legacy solutions and then present an ASP encoding of the following new constraint to capture the Manhattan distance:

C_1. **ManhattanDistance**: The obtained timetable should be as similar as possible to the legacy timetable. Each legacy assignment not in the obtained timetable counts as 1 violation.

A solution of CB-CTT is represented as a set of atoms of the predicate `assigned/4` as can be seen in Listing 6. Each atom `assigned(C,R,D,P)` expresses an assignment such that a lecture of a course `C` is assigned to a room `R` at a period `P` on a day `D`. In a similar way, we represent a legacy solution as a set of atoms of `legacy(assigned(C,R,D,P))`. The *teaspoon* encoding of C_1 is shown in Listing 15. For every course `C`, room `R`, day `D`, and period `P`, the rule generates a penalty atom with the cost of `W` and with the priority level of `Prior` if an assignment represented by `assigned(C,R,D,P)` is satisfied in the legacy solution, but not in the obtained solution.

Our proposed approach can handle at least three types of differences between the original and the new problems:

- Removal version : Remove some constraints from the original problem.
- Additional version : Add some new constraints to the original problem.
- Mixed version : Mix the removal and additional versions.

For illustration, let us consider a simple scenario for an additional version consisting of a problem instance in UD1 as an original problem, its optimal solution as a legacy solution, and the same instance in UD2 as a new problem (S_5. RoomStability is added). In this scenario, the optimization criterion is changed from the original to the new problem due to the additional soft constraint S_5. Therefore, the optimization quality (viz. the penalty of soft constraints) might be different between them depending on the instance. The stabilization quality (viz. the change from the legacy solution) is the number of changed assignments. Note that the number of possible changes is identical to the size of the legacy solution.

ASP facts representing the UD2 formulation plus the Manhattan distance C_1 are shown in Listing 16. The constants `p1` and `p2` represent the priority levels of soft constraints in lexicographic optimization. We can obtain two extreme Pareto optimal solutions by varying those priority levels. If `p1`>`p2`, one extreme solution prioritizing optimality is computed. If we change the priority level to `p1`<`p2`, another extreme solution prioritizing stability is computed. To be specific, for `comp13` instance, we can obtain two extreme Pareto optimal solutions $(59,185)$ and $(233,0)$. In each solution, the first value is the penalty of soft constraints, and the second is the change from the legacy solution. Since the optimization qualities of `comp13` are 31 in UD1 and 59 in UD2, and the number of possible changes is 308, the extreme solution $(59,185)$ has the minimal penalty 59 but 60% of the legacy assignments are changed. The solution $(233,0)$ has the minimal change 0 but incurs higher penalty 233.

Table 8 Pareto optimal solutions of bi-objective course timetabling problem combining CB-CTT and MPP

Instance	#Possible changes	Pareto optimal solutions	#Optima
`comp02`	283	$(164,0)$	1
`comp04`	286	$(35,172)$, $(197,0)$	2
`comp06`	361	$(225,0)$	1
`comp07`	434	$(246,0)$	1
`comp08`	324	$(223,0)$	1
`comp10`	370	$(206,0)$	1
`comp11`	162	$(0,39)$,$(26,1)$, $(25,2)$, $(24,3)$, $(23,4)$, $(22,5)$, $(21,6)$, $(20,7)$,$(27,0)$	9
`comp13`	308	$(59,185)$, $(232,1)$, $(233,0)$	3
`comp14`	275	$(209,0)$	1
`comp16`	366	$(224,0)$	1
`comp17`	339	$(234,0)$	1
`comp19`	277	$(198,0)$	1
`comp20`	390	$(208,0)$	1
`DDS1`	900	$(562,0)$	1
`DDS2`	146	$(0,48)$, $(47,1)$, $(46,2)$, $(45,3)$, $(48,0)$	5
`DDS3`	206	$(0,104)$, $(96,1)$, $(95,2)$, $(97,0)$	4
`DDS5`	560	$(0,321)$, $(298,1)$, $(299,0)$	3
`DDS6`	324	$(172,0)$	1
`DDS7`	254	$(0,160)$, $(133,1)$, $(132,2)$, $(134,0)$	4
`EA01`	351	$(65,179)$, $(242,1)$, $(243,0)$	3
`EA02`	241	$(0,133)$, $(112,1)$, $(111,2)$, $(113,0)$	4
`EA04`	688	$(0,428)$, $(379,1)$, $(380,0)$	3
`EA05`	275	$(0,84)$, $(82,1)$, $(83,0)$	3
`EA06`	300	$(5,151)$, $(127,1)$, $(128,0)$	3
`EA07`	653	$(0,399)$, $(381,1)$, $(382,0)$	3
`EA08`	486	$(0,251)$, $(225,1)$, $(226,0)$	3
`EA09`	423	$(4,226)$, $(223,1)$, $(224,0)$	3
`EA10`	284	$(0,158)$, $(124,1)$, $(123,2)$, $(125,0)$	4
`EA11`	139	$(0,52)$, $(51,1)$, $(1,51)$, $(50,2)$, $(52,0)$	5
`EA12`	174	$(4,73)$, $(5,72)$, $(76,1)$, $(77,0)$	4
`test2`	223	$(130,0)$	1
`test3`	252	$(200,0)$	1
`toy`	16	$(0,7)$, $(2,3)$, $(4,1)$, $(3,2)$, $(1,5)$, $(5,0)$	6*
`Udine1`	360	$(180,0)$	1
`Udine2`	383	$(8,201)$, $(205,1)$, $(206,0)$	3
`Udine3`	324	$(180,0)$	1
`Udine4`	201	$(64,109)$, $(167,1)$, $(168,0)$	3
`Udine5`	337	$(0,162)$, $(155,1)$, $(156,0)$	3
`Udine6`	329	$(0,156)$, $(148,1)$, $(149,0)$	3
`Udine7`	356	$(0,188)$, $(179,1)$, $(180,0)$	3
`Udine8`	400	$(29,223)$, $(241,1)$, $(242,0)$	3
`Udine9`	312	$(21,166)$, $(180,1)$, $(181,0)$	3

In the following, we explain how to find other Pareto optimal solutions. We iteratively obtain Pareto optimal solutions by alternating between finding lexicographic optima and restricting the objective functions using the values of said optima. The method consists of two steps: (i) calculate lexicographic optimal solutions for `p1`>`p2` or `p2`>`p1`; (ii) if solutions have been found, restrict the search space to only allow for value vectors that are incomparable to previous solutions and return to (i). Since all lexicographic optimal solutions are also Pareto optimal and the solutions we iterate are incomparable by design to previous solutions, all lexicographic optima found in (i) are also Pareto optimal. And also, we are sure to obtain the Pareto front given enough time because we consider the whole search space in the beginning. Reconsidering the example from before, we would obtain lexicographic optima $(59,185)$ and $(233,0)$ in Step (i), then we restrict the search space by adding constraints forcing optimality to be between 59 and 233 and stability to be between 0 and 185. Then, we are able to find another Pareto optimal solution $(232,1)$ via lexicographic optimization for `p2`>`p1` in the restricted search space.

We implemented the proposed method for solving bi-objective course timetabling problems combining CB-CTT and MPP by using multi-shot ASP solving techniques and the built-in lexicographic optimization in *clingo*. Our empirical analysis considers the same scenario as discussed above. We used 42 instances in Table 4 for which optimal solutions in UD1 were found. The computational environment is the same as in Section 6. As configuration of *clingo*, we chose *USC11-JP* since it was the best 1-way configuration in the single-objective optimization experiments.

Table 8 shows the Pareto optimal solutions that we were able to obtain. The first column contains the instance name. The second column displays the total number of possible changes. The third and forth columns show a set of Pareto optimal solutions and its size respectively. The third column is separated into three parts. The leftmost entry gives the extreme Pareto optimal solution regarding optimality, the rightmost entry regarding stability, and in between are the additional Pareto optimal solutions. A star '$*$' attached to the number of optimal solutions indicates that the Pareto front has been found.

We were able to calculate the extreme Pareto optimal solutions for `p2`>`p1` for all 42 instances and the extreme Pareto optimal solutions for `p1`>`p2` for 26 instances. Finding the former is trivial since it amounts to calculating the optimization quality of the legacy solution taking the additional soft constraint S_5 into consideration. The latter requires a more involved optimization, first optimizing according to the soft constraints in UD2 and then finding the optimal solution with the best stability. For 25 out of the 26 instances, we successfully obtained an additional Pareto optimal solution, and for 9 instances, we computed 4 Pareto optimal solutions or more. We can observe the completeness of our approach on the instance `toy` since we were able to find the Pareto front.

Overall, the results show that we can leverage the efficient lexicographic optimization of *clingo* and the information provided by the vectors of lexicographic optima to successfully compute additional Pareto optimal solutions other than the extreme cases. With that, our system provides a complete method for finding the Pareto front of multi-objective course timetabling problem, as illustrated on the small instance `toy`.

Finally, we discuss some more details of our experimental results from a practical point of view. In our experiments, we first calculated the extreme Pareto optimal solutions regarding stability and optimality giving both a time limit of 3 hours. Afterwards, we used another 3 hours to enumerate additional Pareto optimal solutions. We showed that the proposed method can find a small subset of Pareto optimal solutions for many real-world instances. However, at the same time, we observed that the current implementation is insufficient to find a promising subset that includes solutions such that both criteria are reasonably well-balanced (e.g., (Schwind et al, 2014)). As an example, let us see the result of `DDS2` instance. We obtained five Pareto optimal solutions: $(0,48)$, $(47,1)$, $(46,2)$, $(45,3)$, and $(48,0)$. Again, in each solution, the first value is the penalty of soft constraints, and the second is the change from the legacy solution. Since the optimization quality of `DDS2` is 0 both in UD1 and UD2, and the number of possible changes is 146, the extreme solution $(0,48)$ has the minimal penalty 0 but 33% of the legacy assignments are changed. The extreme solution $(48,0)$ has the minimal change 0 but incurs higher penalty 48 than the optimal quality 0. The others $(47,1)$, $(46,2)$, and $(45,3)$ are additional Pareto optimal solutions, but the improvement from the complete stability $(48,0)$ is small. This shows a limitation of our approach at present. To overcome this issue, there might be at least two approaches. One is spending more times, since our approach is complete, and we can find more Pareto optimal solutions. Another is extending our approach with an advanced technique in MODOP solving based on P-minimal model generation (Soh et al, 2017). It would be promising because P-minimal models can be efficiently computed by ASP.

8 Conclusion

We presented an ASP-based approach for solving the curriculum-based course timetabling (CB-CTT) problems. The resulting system *teaspoon* [13] is built upon general-purpose ASP systems, in our case *clingo*. That is, *teaspoon* relies on high-level ASP encodings presented in Section 4, 5, and 7 and delegates both the grounding and solving tasks to *clingo*.

The main features of our declarative approach are as follows:

Expressiveness. The expressive power of ASP's modelling language enables us to compactly express a wide variety of hard and soft constraints of CB-CTT as demonstrated by a collection of *teaspoon* encodings. Given new constraints (e.g., Manhattan distance), all we have to do is adding ASP rules.

Flexibility. *teaspoon* provides flexible lexicographic optimization and easy composition of different formulations, since any combination of constraints can be represented as ASP facts. Consequently, it enables a timetable keeper to experiment with different formulations as well as to optimize the obtained timetable with different priority levels of soft constraints, at a purely declarative level.

Efficiency. Our empirical analysis considers all instances in five different formulations, which are publicly available from the CB-CTT portal. We have contrasted the performance of *teaspoon* with the best known bounds obtained so far via more dedicated implementations. *teaspoon* demonstrated that our declarative approach allows us to compete with state-of-the-art CB-CTT solving techniques.

Scalability. The optimized encoding drastically reduces the grounding time compared to the basic encoding. Consequently, our declarative approach scales to large instances in complex formulations, as demonstrated by the fact that *teaspoon* was able to find upper bounds for very large instances in the category `erlangen` with every formulation, and 24 of them were unsolvable before.

Extensibility. The high-level approach of ASP facilitates extensions and variations of first-order encodings. From this viewpoint, we extended the *teaspoon* system to solving multi-objective course timetabling problem combining CB-CTT and minimal perturbation problems with two criteria of optimality and stability.

Perhaps the most relevant related works are problem encodings in Integer Programming (Burke et al, 2010a,b, 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/3` and/or `assigned/4` respectively. SAT/MaxSAT encodings (Achá and Nieuwenhuis, 2012) 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.

Our ASP-based approach can be applied to a wide range of timetabling problems such as *school timetabling*, *examination timetabling*, and *post-enrollment course timetabling*. Multi-objective optimization implemented in *teaspoon* can be further extended in selecting a promising subset of Pareto optimal solutions by using an advanced technique in MODOP solving based on P-minimal model generation (Soh et al, 2017). ASP-based large neighborhood search (LNS) for course timetabling can be promising because a MaxSAT-based LNS has been recently shown to be effective for high school timetabling (Demirovic and Musliu, 2017). For this, we developed a prototype implementing a simple neighborhood search using multi-shot ASP solving with *teaspoon*. In a preliminary experiment on some `comp` instances

[13] All source code is available from `https://potassco.org/doc/apps/`

in UD5, the prototype was able to find new bounds for 33 for `comp10`, 135 for `comp13`, 142 for `comp09`, and 143 for `comp21`. We will investigate these possibilities, and the results will be applied to representing and reasoning more practical timetabling problems.

References

Abdullah S, Turabieh H, McCollum B, McMullan P (2012) A hybrid metaheuristic approach to the university course timetabling problem. Journal of Heuristics 18(1):1–23

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

Andres B, Kaufmann B, Matheis O, Schaub T (2012) Unsatisfiability-based optimization in clasp. In: Dovier A, Santos Costa V (eds) Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12), Leibniz International Proceedings in Informatics (LIPIcs), vol 17, pp 212–221

Ansótegui C, Bonet M, Levy J (2013) SAT-based MaxSAT algorithms. Artificial Intelligence 196:77–105

Banbara M, Soh T, Tamura N, Inoue K, Schaub T (2013) Answer set programming as a modeling language for course timetabling. Theory and Practice of Logic Programming 13(4-5):783–798

Banbara M, Inoue K, Kaufmann B, Schaub T, Soh T, Tamura N, Wanko P (2016) teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. In: Burke EK, Di Gaspero L, Özcan E, McCollum B, Schaerf A (eds) Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling (PATAT'16), pp 13–32

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

Barták R, Müller T, Rudová H (2004) A new approach to modeling and solving minimal perturbation problems. In: Apt KR, Fages F, Rossi F, Szeredi P, Váncza J (eds) Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, (CSCLP'03), Springer, LNCS, vol 3010, pp 233–249

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

Biere A, Heule M, van Maaren H, Walsh T (eds) (2009) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol 185. IOS Press

Bonutti A, De Cesco F, Di Gaspero L, 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 EK, Petrovic S (2002) Recent research directions in automated timetabling. European Journal of Operational Research 140(2):266–280

Burke EK, Marecek J, Parkes AJ, Rudová H (2010a) Decomposition, reformulation, and diving in university course timetabling. Computers & Operations Research 37(3):582–597

Burke EK, Marecek J, Parkes AJ, Rudová H (2010b) A supernodal formulation of vertex colouring with applications in course timetabling. Annals of Operations Research 179(1):105–130

Burke EK, Marecek J, Parkes AJ, Rudová H (2012) A branch-and-cut procedure for the Udine course timetabling problem. Annals of Operations Research 194(1):71–87

Demirovic E, Musliu N (2017) MaxSAT-based large neighborhood search for high school timetabling. Computers & OR 78:172–180

Di Gaspero L, Schaerf A (2003) Multi-neighbourhood local search with application to course timetabling. In: Burke EK, Causmaecker PD (eds) Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT'02), Springer, Berlin Heidelberg, LNCS, vol 2740, pp 262–275

Di Gaspero L, Schaerf A (2006) Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modelling and Algorithms 5(1):65–89

Di Gaspero L, McCollum B, Schaerf A (2007) The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical report, Queen's University, Belfast, United Kingdom

Eén N, Sörensson N (2003) Temporal induction by incremental SAT solving. Electronic Notes in Theoretical Computer Science 89(4):543–560

Ehrgott M (2005) Multicriteria Optimization. Springer Berlin Heidelberg

Erdem E, Gelfond M, Leone N (2016) Applications of ASP. AI Magazine 37(3):53–68

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

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

Gebser M, Kaufmann B, Otero R, Romero J, Schaub T, Wanko P (2013) Domain-specific heuristics in answer set programming. In: desJardins M, Littman M (eds) Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI'13), AAAI Press, pp 350–356

Gebser M, Kaminski R, Kaufmann B, Romero J, Schaub T (2015a) Progress in clasp series 3. In: Calimeri F, Ianni G, Truszczyński M (eds) Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15), Springer-Verlag, LNAI, vol 9345, pp 368–383

Gebser M, Kaminski R, Obermeier P, Schaub T (2015b) Ricochet robots reloaded: A case-study in multi-shot ASP solving. In: Eiter T, Strass H, Truszczyński M, Woltran S (eds) Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation: Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday, Springer-Verlag, LNAI, vol 9060, pp 17–32

Geiger MJ (2012) 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

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

Hutter F, Hoos H, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11), Springer-Verlag, LNCS, vol 6683, pp 507–523

Lach G, Lübbecke ME (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

Lü Z, Hao JK (2010) Adaptive tabu search for course timetabling. European Journal of Operational Research 200(1):235–244

Marler R, Arora J (2004) Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 26(6):369–395

Marques-Silva J, Planes J (2007) On using unsatisfiability for solving maximum satisfiability. CoRR abs/0712.1097

McCollum B (2007) A perspective on bridging the gap between theory and practice in university timetabling. In: Burke EK, Rudová H (eds) Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06), Revised Selected Papers, Springer, LNCS, vol 3867, pp 3–23

McCollum B, Schaerf A, Paechter B, McMullan P, Lewis R, Parkes AJ, Di Gaspero L, Qu R, Burke EK (2010) Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing 22(1):120–130

Müller T (2009) ITC2007 solver description: a hybrid approach. Annals of Operations Research 172(1):429–446

Müller T, Rudová H, Barták R (2005) Minimal perturbation problem in course timetabling. In: Burke EK, Trick MA (eds) Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT'04), Springer, LNCS, vol 3616, pp 126–146

Narodytska N, Bacchus F (2014) Maximum satisfiability using core-guided MaxSAT resolution. In: Brodley C, Stone P (eds) Proceedings of the Twenty-Eighth National Conference on Artificial Intelligence (AAAI'14), AAAI Press, pp 2717–2723

Niemelä I (1999) Logic programs with stable model semantics as a constraint programming paradigm. Ann Mathematics and Artificial Intelligence 25(3–4):241–273

Phillips AE, Walker CG, Ehrgott M, Ryan DM (2016) Integer programming for minimal perturbation problems in university course timetabling. Annals of Operations Research pp 1–22

Rudová H, Müller T, Murray KS (2011) Complex university course timetabling. Journal of Scheduling 14(2):187–207

Sakkout HE, Wallace M (2000) Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 4(5):359–388

Schaerf A (1999) A survey of automated timetabling. Artificial Intelligence Review 13(2):87–127

Schwind N, Okimoto T, Konieczny S, Wack M, Inoue K (2014) Utilitarian and egalitarian solutions for multi-objective constraint optimization. In: Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'14), IEEE Computer Society, pp 170–177

Sinz C (2005) Towards an optimal CNF encoding of Boolean cardinality constraints. In: van Beek P (ed) Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05), Springer-Verlag, LNCS, vol 3709, pp 827–831

Soh T, Banbara M, Tamura N, Berre DL (2017) Solving multiobjective discrete optimization problems with propositional minimal model generation. In: Beck JC (ed) Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming (CP'17), Springer, LNCS, vol 10416, pp 596–614

Zivan R, Grubshtein A, Meisels A (2011) Hybrid search for minimal perturbation in dynamic CSPs. Constraints 16(3):228–249