184.701 UE Deklaratives Problemlösen SS 2017

Exercise 3: Advanced Answer-Set Programming

Uwe Egly, Hans Tompits, and Christoph Redl

Institut für Informationssysteme,
Arbeitsbereich Wissensbasierte Systeme 184/3,
Technische Universität Wien

Deadline First Turn-In: 11.06.2017 23:55
Deadline Second Turn-In: 25.06.2017 23:55

1 Introduction

In this exercise we will extend the previous problem of *assigning teachers to classes* and delve into the various uses of aggregates and optimization. The task consists of constructing a suitable *extended logic program* (ELP) according to the *Guess-and-Check* paradigm such that the solutions of the given problem are determined by the answer sets of the program. However, besides default negation, disjunction, and integrity constraints, this time it is mandatory to use *aggregate atoms* and *cardinality constraints* as formalization tools. Moreover, extensions like *weight constraints* and *optimize statements* shall be deployed to compute optimal solutions. The evaluation of the constructed programs shall be carried out with the Potassco tools clasp 3.1.2 and gringo 4.5.0, see

`http://potassco.sourceforge.net/`.

Encode your logic programs in files with suffix `.lp`.

We further want to recommend the usage of the answer set programming IDE `SeaLion`, developed at our department. It supports both DLV and Potassco including useful features such as syntax highlighting, visualisation of answer sets, and a debugging system. `SeaLion` is realised as a plugin for the `Eclipse` platform and is available from the website

`http://www.sealion.at/`

where we offer standalone versions as well as instructions on how to get `SeaLion` via Eclipse's update mechanism. The `SeaLion` plugin `Kara` (also available in the standalone version and the update site) allows for visualising answer sets. We provide an example visualisation program (an answer-set program specifying how an answer set should be visualised) that is tailored towards the example in the exercise, available at TUWEL. If you want to visualise an answer set, open its context menu in `SeaLion`'s interpretation view, select "Visualisation" and choose the visualisation program available from TUWEL.

2 General problem description

In this exercise, we continue working on the problem that has been addressed in the previous exercise (exercise 2), i.e., the problem of assigning teachers to courses in var-

ious classes. This time, we deal with slightly different constraints for a more realistic setting. We will first extend the teacher assignment problem, allowing that the work in a single course of a class be divided among multiple teachers. Then, we will formalize *soft requirements*, which ought to be satisfied as much as possible, eventually turning the problem into an optimization task.

3 Subtask 0

3.1 Converting to the Potassco input format

For the previous exercise you created two programs for use with the DLV system. As we will be using Potassco for this exercise and naturally would like to build upon our existing work, the first task consists of porting your two files from the DLV input format to the format used by the Potassco suite.[1] For doing that, use choice rules instead of disjunctive rules.

This process should result in two files `check.lp` and `guess.lp`, namely the encoding of the guess and check programs for the problem described in exercise 2. In order to verify the correctness of your conversion you can use the official test cases for exercise 2 (as provided via TUWEL) and compare the results of the DLV system with those of the Potassco suite.

4 Subtask 1

4.1 Modifying the Check program

In the first part of the task, modify your existing programs `check.lp` and `guess.lp` to fulfill the following hard requirements.

First of all, **remove** the following constraints from `check.lp`:

(a_3) Each class should be taught each course by at most one teacher.

(a_7) No teacher should be required to spend more than 24 hours preparing lessons and extra-curricular activities in total every week, where each lesson for grades 1 and 2 accounts for 1 hour of preparation time and each lesson for grades 3 and 4 accounts for 2 hours of preparation time.

(e_2) No teacher should spend more time preparing extra-curricular activities than regular lessons.

In this task, we will allow multiple teachers to be assigned to teach a course in a class. For instance, a class requiring 6 lessons a week in some course might be taught by two teachers, each assigned for 3 lessons a week in that course to that class. To represent the number of lessons assigned to a teacher exchange the predicate `assigned(T,C,S)` with `assigned(T,C,S,L)`, where $L > 0$ represents the number of lessons a teacher `T` teaches to class `C` in course `S`.

Furthermore, **add** these new constraints to `check.lp`:

[1] For more information on the input format to Gringo (Potassco's grounder for answer set programs) see this helpful guide: `http://sourceforge.net/projects/potassco/files/guide/2.0/guide-2.0.pdf/download`

(a_8) No teacher is assigned to teach more than 20 lessons a week. [2]

(e_3) No teacher is assigned to spend more than 10 hours a week for extra-curricular activities.

(e_4) Each teacher should spend at most as much time preparing extra-curricular activities as teaching lessons.

4.2 Modifying the Guess program

Given the new requirement of possibly assigning multiple teachers to a course in any class, we also have to modify `guess.lp`.

Replace the requirement

(g_1) for each course requiring $L > 0$ lessons in the curriculum of the grade of a class, exactly one teacher is assigned to teach that course in that class;

by

(g_1') for every course in a class that, according to the curriculum of its grade, requires $L > 0$ lessons a week in that course, **at least one** teacher is assigned to teach L_2 ($0 < L_2 \leq L$) lessons;

Moreover, ensure that the following constraints are **satisfied**:

(g_4) In each class requiring $L \leq 3$ lessons a week in a course, no more than one teacher is assigned to that course;

(g_5) Whenever multiple teachers are assigned to a single course of a class, each of those teachers must be assigned for $L \geq 2$ lessons a week to that course in the class;

(g_6) To each course required in a class, a teacher can only be assigned once (i.e., there should not be multiple assignments of the same teacher to the same class in the same course featuring different numbers of lessons);

(g_7) For each class and each course, the sum of lessons of the course taught by different teachers in that class must be equal to the required number of lessons, which is given by the curriculum for the grade of the class.

5 Subtask 2

In the next part we will start taking preferences of teachers and classes into account. Dispositions of teachers towards courses and dispositions of classes towards teachers are expressed as integral values on a scale from 1 to 3, meaning:

1: "I do not like to teach that course" / "We do not like to be taught by that teacher."

2: "I am / We are indifferent about this course / teacher."

3: "I would like to teach that course" / "We would like to be taught by that teacher."

[2] Note that this is different from — i.e. simpler than — the notion of preparation for lessons presented in exercise 2.

Actual preferences will be given as facts of the predicate `preference(A,B,V)`, where `A` and `B` are either a teacher and a course or a class and a teacher, and `V` is an element of $1, 2, 3$. For instance, a teacher `t` might be in favor of teaching a course `s`, which would be expressed as `preference(t,s,3)`. The same predicate is used for classes' dispositions towards teachers, e.g., a class `c` might dislike a teacher `t`, in which case the respective fact would be `preference(c,t,1)`.

5.1 Complementing the Check program with preferences

Several new requirements are introduced along with these preferences. Realize them in a **separate** file `check-preferences.lp` using hard constraints.

(p_1) No teacher gives more than one preference for a course. The value of that preference must be 1, 2 or 3.

(p_2) No class gives more than one preference for a teacher. The value of that preference must be 1, 2 or 3.

(p_3) When a teacher or class `A` does not give her disposition towards a course or teacher `B`, respectively, assume it to be neutral (i.e., `preference(A,B,2)` must be present in all answer sets).

(p_4) No teacher should give negative preferences for more than half of all courses. That is, in case there are n different courses, each teacher may give negative preferences for at most $\lfloor n/2 \rfloor$ different courses.

(p_5) No class gives more negative preferences as it gives positive preferences to a teacher.

Finally, implement the following rule in `check-preferences.lp`:

(p_6) In order to determine the degree of compatibility of each teacher `T` paired with each class `C`, we deduce facts of predicate `rating(T,C,V)` for each such pair. `V` is the result of

- the sum of the products of lessons `L` for each course `S` that teacher `T` has been assigned to in class `C` multiplied by teacher `T`'s disposition towards that course (informally speaking: the lessons a week a teacher has been assigned in all courses in a class, weighted by the teacher's preference for each such course)
- multiplied by class `C`'s disposition towards teacher `T`.

For instance, consider a teacher `t`, who has been assigned to teach a class `c` in courses `s1` and `s2`, where both require $L = 4$ lessons a week, and the teacher has been assigned to all of them. Teacher `t` gave positive and neutral preferences for these courses: `preference(t,s1,3). preference(t,s2,2).` Class `c` gave a negative preference for teacher `t`: `preference(c,t,1).` Then the rating of that teacher-class pairing equals $(4 \cdot 3 + 4 \cdot 2) \cdot 1 = 20$ and consequently `rating(t,c,20)` should be included in each answer set.

5.2 Testing the Check-Preferences program

Test the functionality of `check-preferences.lp` on its own, i.e. without using any of the files `guess.lp` or `check.lp`, by constructing at least **five test cases**. To this end, for each test case n ($1 \leq n \leq 5$), safe your data in files as specified in exercise 2, i.e. `courses`n`.facts`, `classes`n`.facts`, `teachers`n`.facts` and `asgn`n`.facts`. Information on teachers' preferences is stored in `teachers`n`.facts` using facts of the form `preference(T,S,V)`, where `T` is a teacher, `S` is a course, and `V` is one of 1, 2 or 3. Information on class preferences is stored in `classes`n`.facts` and uses facts of the form `preference(C,T,V)`, where `C` is a class, `T` is a teacher and `V` is one of 1, 2 or 3. If you specify more than five test cases, use $n \geq 9$ for the additional file names (since $6 \leq n \leq 8$ are used for the mandatory test cases of subtask 3).

Exercise Requirement: Make sure `check-preferences.lp` is self-contained, i.e., define *all* auxiliary predicates that you use in this file. However, do not define predicates that are part of the input there. Thus, if you want to re-use auxiliary predicates defined in `guess.lp` or `check.lp`, copy their definitions to the file `check-preferences.lp` (but *do not* copy constraints or guessing rules).

6 Subtask 3

Apart from the hard requirements accounted for by `guess.lp`, `check.lp`, and `check-preferences.lp`, there are soft requirements for the teacher assignment problem that shall be respected as much as possible. This subtask consists of implementing such optimizations in a **separate** file named `optimize.lp`.

6.1 Solving the Optimization problem

Consider the following soft requirements for an optimal assignment of teachers to courses in classes:

(o_1) Minimize the deviation of the number of weekly hours spent (for teaching lessons and preparing extra-curricular activities) by any teacher from the average.

(o_2) Optimize the choice of paired teachers and classes by maximizing the cumulative value of `rating/3`.

(o_3) Maximize teachers' number of weekly lessons at their main classes.

The above requirements are given in order of decreasing priority, meaning that any solution must be optimal in respect to (o_1) before considering requirement (o_2) and the same applies to (o_2) and (o_3).

Exercise Requirements:

ad (o_1) In order to formalize condition (o_1), compute the number of hours spent (for teaching lessons and preparing extra-curricular activities) per teacher, averaged over all teachers. Then use the difference of each teacher's number of spent hours to the average number as a penalty for the optimization.
For simplicity, we define the average a of the values $\{v_1, ..., v_n\}$ as the sum of these values $S_{val} = \sum_{i=1}^{n} v_i$ divided by the number of values n. Then, a is the greatest integer such that $a \cdot n \leq S_{val}$.

ad (o_2) For (o_2), maximize the total value of all rated pairs of teachers and classes with respect to the value `V` in `rating(T,C,V)`.

ad (o_3) Concerning (o_3), define a predicate `mainClass(T,C)` which indicates that for a teacher `T`, class `C` is her main class. Each teacher's single main class is determined by the following set of rules:

- if a teacher is the form teacher of a class, that class is her main class;
- in case that a teacher is not form teacher of any class, the main class is the class of the highest grade that she is teaching. If this leaves a tie between two or multiple classes, the classes are ordered by their name and the least one (with respect to the built-in < operator) is chosen as the main class of the teacher.

Then, use the number of assigned lessons in classes besides the main class as a penalty for the assignment.

Realize the optimization according to the soft requirements defined above and implement them in a file `optimize.lp`.

6.2 Testing the Optimization program

Test the functionality of the program `optimize.lp` on its own, i.e., without using the files `guess.lp`, `check.lp` and `check-preferences.lp`, by constructing **three more test cases**. For each test case n ($6 \leq n \leq 8$), provide the input data in files following the same naming conventions as before. If you specify more than three test cases, then use $n \geq 9$ for additional file names.

Note: It is *not* required that the input data specified in `asgn`n`.facts` for testing the optimization program `optimize.lp` consists of facts only: you may find it useful to also use choice rules (with head predicate `assigned`) in order to specify input alternatives.

Exercise Requirement: Again, `optimize.lp` has to be self-contained. In case you do re-use auxiliary predicates from one of the files `guess.lp`, `check.lp` or `check-preferences.lp`, make sure to define them in `optimize.lp` as well (copy their definition), otherwise the latter may not work as intended when tested stand-alone (But *do not* copy constraints or guessing rules).

Important:

1. Be sure that you **follow the naming conventions** as pointed out above.
2. All of the files mentioned above should be contained in **one directory**!

7 Submission

Upload a ZIP-file with your solution to the TUWEL system. Before the deadline you can upload your solution several times. Submissions after the deadline will be rejected. Use the target

```
make zip
```

from the `Makefile` to create your ZIP-file. The result will be `XXXXXXX_ex3.zip`, where `XXXXXXX` is replaced by your immatriculation number (make sure you have set the `MNR` variable in the `Makefile` accordingly).

In case of questions, please post a message to the official TISS forums or send an email to

`dps-2017s@kr.tuwien.ac.at`.