VL Logikorientierte Programmierung

Answer-Set Programming: Exercise 2

Uwe Egly, Michael Fink, and Hans Tompits

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

Summer term 2009

1 Introduction

In this exercise, search and optimization problems, again in the context of *scientific conference organization*, shall be solved. Once more, the task is to construct, for a given problem, 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 also allowed (and encouraged) to use *aggregate functions* as formalization tools. Moreover, *weak constraints* shall be deployed to compute optimal solutions (*Guess-Check-Optimize*) using DLV (`http://www.dlvsystem.com`, resp., installed in the lab).

2 General Problem Description

In this exercise, we continue working on the referee problem that has been addressed in the previous exercise (Answer-Set Programming Exercise 1), i.e., the problem of assigning papers submitted to a conference to members of the PC. For many conferences, prior to the assignment of submissions to PC-members, there is a *bidding phase*, where PC-members can bid on papers. A bid expresses a degree of preference of a member of the PC for a concrete submitted paper. We will first extend the referee problem taking bids of PC-members into account, specifying some further *hard requirements* that a valid assignment has to obey. Then, we will formalize some *soft requirements*, which should be satisfied as much as possible, eventually turning the problem into an optimization task, which we will refer to as the *optimal referee problem*.

3 Subtask 1

The first task is to compute solutions to the referee problem as specified in the previous exercise (Answer-Set Programming Exercise 1), additionally taking into account bids of the PC-members. More precisely:

- Each member of the PC has the possibility to *bid* on papers prior to the actual assignment, expressing a degree of preference for a concrete paper. We will assume that bids are integers in the range of $0, \ldots, 3$ with the following meanings:

  0: “I cannot review this paper”,

  1: “I don’t want to review this paper”,

  2: “I am willing to review this paper”,

  3: “I really want to review this paper”.

- The following predicate is used to specify bids, given as input data:

  `bid(M,P,B)`: PC-member `M`'s bid on paper `P` is `B`, where `B` is a constant from $\{0, 1, 2, 3\}$.

3.1 Solving the Search Problem: Guess-and-Check

First of all, write a Guess-and-Check program `guess-and-check.dl` which computes solutions to the referee problem of the previous exercise (Answer-Set Programming Exercise 1). However, rather than just copying your previous solution (`check.dl` and `guess.dl`) into a single file, *use aggregate atoms* in order to simplify your previous solution. Of course you can reuse your test files for the overall program of the previous excercise (cf. Section 4.2 of Answer-Set Programming Exercise 1) to check whether the modified program `guess-and-check.dl` is correct.

Furthermore, encode the following additional hard requirements into a file `check-bids.dl`:

(h1) Each PC-member bids at most once for a paper.

(h2) Each PC-member bids on at least two papers different than the default bid, which is "I don't want to review this paper" (see also (a1) below).

(h3) If a PC-member rates a submission with "I cannot review this paper", this paper must not be assigned to that PC-member.

(h4) A situation where a submission is assigned only to PC-members who rated that paper with "I don't want to review this paper" would be highly unfair and shall be explicitly excluded.

(h5) The assignment average for any PC-member who has been assigned at least one paper is greater than 1.5. I.e., the value of the second argument of `ass_avg` (see below) is greater than 15, and *the first argument is a PC-member*. (To check whether the first argument is a PC-member is necessary for correctness of the overall program, because in Subtask 2 the assignment average will be computed also for submitted papers using the binary predicate `ass_avg`.)

In addition to checking for these hard requirements, complete the provided information on bids by formalizing the following default assumption:

(a1) Any member of the PC that did not bid on a particular submitted paper, is assumed to not wanting to review this paper (bid 1, "I don't want to review this paper").

In order to compute the assignment average for a PC-member proceed as follows. Define an auxiliary predicate `idiv` for integer division first:

(a2) `idiv(X,Y,Z)`: `X`, `Y`, and `Z` are integers, `Y`> 0, such that `Z` is the integer result (rounded down real result) of the division of `X` by `Y` (i.e., `Z` is the greatest integer less or equal the real result of the division).

For instance, `idiv(9,3,3)` and `idiv(40,3,13)` should be true in any answer set.

Hint: Given integers X and Y, the result of the integer division of X by Y is the maximal integer Z, such that $P \leq X$ holds for the product $P = Z \times Y$.

Based on this, define an auxiliary predicate `ass_avg` for the assignment average of a PC-member. This value should *only* be computed *for PC-members who have been assigned at least one paper*. Moreover, to allow for a more fine-grained distinction, i.e., for being able to consider also the first position after the decimal point of the real average value, the bids are scaled by a factor of 10:

(a3) `ass_avg(M,A)`: `A` is the rounded down average of scaled bids issued by PC-member `M` for the *assigned* papers (i.e., `A` is the greatest integer less or equal to the real average value multiplied by 10).

For example, consider a PC-member `m` who has been assigned three papers `p1`, `p2`, and `p3`. The PC-member did bid for these papers as follows: `bid(m,p1,1)`, `bid(m,p2,2)`, and `bid(m,p3,1)`. Then, the sum of these bids is 4 and the sum of scaled bids is $4 \times 10 = 40$. Since the number of assigned papers is 3, we obtain $1.\dot{3}$ as the real average value, and $13.\dot{3}$ as the real average value of the scaled bids. Hence, the assignment average for `m` is 13 (`ass_avg(m,13)` should be true in the answer set).

Hint: Let S be the sum of bids for papers assigned to a PC-member `M`, let $S' = S \times 10$, and let N be the number of papers assigned to `M`. Then, the assignment average value `A` for `M` is `idiv(S',N,A)`.

Exercise Requirement: It is *obligatory to use aggregate atoms* to define the auxiliary predicates `idiv` and `ass_avg`. (Apart from that you are encouraged to use aggregate atoms where suitable, but you are not obliged to do so.)

Hint: Use additional auxiliary predicates for defining the sum of bids for the papers assigned to a PC-member, as well as for determining the number of assigned papers.

3.2 Testing the Guess-and-Check Program

Test the functionality of `check-bids.dl` stand-alone, i.e., without using the file `guess-and-check.dl`, 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 Answer-Set Programming Exercise 1, i.e., `kw`n`.dl`, `submissions`n`.dl`, `pc`n`.dl`, and `asgn`n`.dl`. The additional input on bids (facts of the form `bid(M,P,B)`) should be contained in `pc`n`.dl`. If you specify more than five test cases, then use $n >= 10$ for the additional file names (since $6 \leq n \leq 9$ is used for mandatory test cases of Subtask 2).

Note: When testing `check-bids.dl` stand-alone, the maximal range of integers needed to compute correct solutions depends on the input data (and your implementation). Specifying a range which is too small to carry out all relevant integer calculations may result in answer sets, which do not correspond to correct solutions.

Exercise Requirement: Design the test cases in such a way that specifying a maximal integer value of 120 is sufficient for computing correct solutions according to your implementation. (E.g., for at most four papers assigned to a PC-member, since the maximum bid is 3, $4 \times 3 \times 10 = 120$ is an upper bound for the sum of scaled bids for the papers assigned to a PC-member.)

Exercise Requirement: Make `check-bids.dl` self-contained, i.e., define *all* auxiliary predicates you use in this file. However, do not define predicates here, which are part of the input. So if you want to re-use auxiliary predicates defined in `guess-and-check.dl`, then copy their definition to `check-bids.dl` (but *do not* copy constraints or guessing rules).

4 Subtask 2

Apart from the hard requirements accounted for by `guess-and-check.dl` and `check-bids.dl`, there also exist soft requirements for the referee problem that shall be respected as much as possible. The second task of this exercise is to implement such optimizations (in a file `optimize.dl`).

4.1 Solving the Optimization Problem: Optimize

Consider the following soft requirements for an optimal assignment of submissions to PC-members:

(o1) Maximize the assignment for papers wrt. bids of PC-members (the willingness of assigned PC-members to review the paper).

(o2) Minimize the overlap of reviewers (shared papers).

(o3) Maximize the assignment wrt. bids for PC-members providing high bids (concretized below).

(o4) Maximize the diversity wrt. keywords (fields of competence) for PC-members.

All these requirements have different (decreasing) priorities, where (o1) is the most important and (o4) is considered to be the least important requirement.

Exercise Requirements:

ad (o1) In order to formalize condition (o1), compute the assignment average value for a paper `P` by defining an auxiliary predicate `ass_avg(P,A)` (as for PC-members in the previous subtask, but the first argument is a paper). Furthermore, use the maximum bid issued by *any* PC-member for `P`, and scale it by factor 10 to get an upper bound for the assignment average value of `P`. (Note that `P` need not necessarily have been assigned to a PC-member who issued a maximum bid for it.) For every paper `P`, assign the difference between the upper bound and the assignment average value as a penalty for the assignment.

As an example, consider a paper `p` with an assignment average value of 13 (that is `ass_avg(p,13)` holds) for which some PC-member `m` did bid 2 (`bid(m,p,2)`), and this is also the highest bid for `p`. Then, this paper contributes a penalty of $2 \times 10 - 13 = 7$ to (o1) for the current assignment.

Remark: Do not rely on (*) in `guess-and-check.dl`, i.e., that each paper is assigned to exactly four PC-members, for computing `ass_avg`. Rather count the number of PC-members assigned to a paper in order to make `optimize.dl` reusable in other settings as well.

ad (o2) Concerning (o2), define an auxiliary predicate `overlap(P1,P2,O)` yielding an overlap value `O` for a pair of papers `P1` and `P2` as follows: `O` is the product of the number of PC-members assigned to both papers and a factor f, where $f = 0$ if `P1` $=$ `P2`, $f = 1$ if `P1` and `P2` do not have an author in common, and $f = 2$ if `P1` and `P2` are (co-)authored by a common author. Every paper incurs a penalty for the assignment that is given by the sum of the overlap values with all other papers.

For instance, consider a paper `p1` that has three PC-members assigned which are also assigned to paper `p2`, and two PC-members assigned to `p1` are assigned to a paper `p3`, as well. Moreover, `p1` and `p3` have an author in common, whereas `p1` and `p2` do not. Then, `p1` causes a penalty for (o2) of $3 \times 1 + 2 \times 2 = 7$.

ad (o3) Every PC-member who has at least one paper assigned and issued two or more bids 3 ("I really want to review this paper"), but has an assignment average less than 2.0 (`ass_avg` value less than 20) causes a penalty for (o3) given by the difference between 20 and its `ass_avg` value.

ad (o4) For every PC-member `M`, the number of keywords associated to `M` such that no paper with this keyword is assigned to `M` is the penalty contributed to (o4) by `M`.

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

4.2 Testing the Optimization Program

Test the functionality of the program `optimize.dl` stand-alone, i.e., without using the files `guess-and-check.dl` and `check-bids.dl`, by constructing **four more test cases**. For each test case n ($6 \leq n \leq 9$), provide the input data in files following the same naming convention as before. If you specify more than four test cases, then use $n >= 10$ for the additional file names.

Note: When testing `optimize.dl` stand-alone, the maximal range of integers needed to compute correct solutions depends on the input data and the way you implemented the requirements for `optimize.dl`. Specifying a range which is too small to carry out all relevant integer calculations may result in answer sets, which do not correspond to correct solutions.

Exercise Requirement: Design the test cases in such a way that specifying a maximal integer value of 120 is sufficient for computing correct solutions according to your implementation.

Exercise Requirement: Also `optimize.dl` has to be self-contained. In case you do reuse auxiliary predicates from `guess-and-check.dl` or `check-bids.dl`, make sure to define them in `optimize.dl` 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.)

5 Subtask 3

5.1 Testing the Overall Program: Guess-Check-Optimize

In order to test the overall program solving the optimal referee problem—which consists of `guess-and-check.dl`, `check-bids.dl` and `optimize.dl`—download the conference data given by the archive `conference.tar.gz` (containing `kw.dl`, `submissions.dl`, and `pc.dl`) from TUWEL, and answer the following questions (fill in the form that will be available in TUWEL):

1. How many optimal referee assignments exist for this conference?
2. To which PC-members is paper `p2` assigned in optimal assignments?
3. What is (are) the assignment average(s) of PC-member `m3` in optimal assignments?
4. What is (are) the assignment average(s) of paper `p1` in optimal assignments?
5. What is (are) the overlap value(s) of papers `p2` and `p3` in optimal assignments?
6. Is there a violation of (o3) in optimal assignments, and if so, which PC-members are affected?

Hint: You may find the DLV command-line options `-filter` and `-pfilter` useful for answering (most of) these questions.

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**!
3. Do not include the command #`maxint` in your files—rather, execute DLV with the corresponding command-line option `-N` to specify the known numbers.

In case of questions, please ask the teaching assistants or send an email to

`logprog-fragen-ss09@kr.tuwien.ac.at`.