ASP introduction and pseudo-perturbations maximization program

1 ASP introduction

Answer Set Programming (ASP) [1] is a declarative programming paradigm used to specify knowledge bases in the form of rules. Programs are built from atoms, which consist of a predicate with arguments that are terms. The simplest types of terms are integers, constants, and variables. Constants and variables are distinguished by their first letter, respectively, a lower case and an upper case. Atoms are elementary constructions for representing knowledge and are used to make simple verbal assertions.

Rules in ASP consist of a *head* and a *body*, where the head, composed of a set of atoms, specifies the goal or conclusion to be derived, and the body specifies the conditions under which the conclusion is true. *Literals*, which can be positive or negative, represent propositions in the body of a rule.

The rules of an ASP program are as follows:

- Fact: H.
- Rule: $H \leftarrow L_1, ..., L_n$.
- Integrity constraint: $\leftarrow L_1, ..., L_n$.

where $H, L_1, ..., L_n$ are literals, part of the head and the body. Each L_i is a literal of the form `A` or `not A`, where `A` is an atom and the logical connector `not` is the default negation. We say that a literal L is positive if it is an atom (`A`) and negative otherwise (`not A`).

Facts in ASP are rules with an empty body and only a head. They are principally used to represent knowledge. Integrity constraints are used in ASP to specify restrictions on the possible interpretations of a knowledge base, ensuring that only valid models are considered.

We introduce a small logic program example, illustrating concepts seen below (see Listing 1).

```
1  cell(c1). cell(c2). cell(c3).
2  class(early_TE). class(medium_TE). class(late_TE).
3  be_part(c1,early_TE). be_part(c2,medium_TE). be_part(c3,late_TE).
4  gene(g1). gene(g2).
5  expr(c1,g1,0). expr(c1,g2,0).
6  expr(c2,g1,0). expr(c2,g2,1).
7  expr(c3,g1,1). expr(c3,g2,1).
8  pert(C,G,S,CL) :- expr(C,G,S), cell(C), gene(G), be_part(C,CL).
9  {sel_pert(C,G,S,CL) : pert(C,G,S,CL)}.
10 :- sel_pert(_,_,_,early_TE).
11 #show sel_pert/4.
```

Listing 1: Example of a logic program.

This logic program defines knowledge via the facts on lines 1-7. For example, the predicate `expr(c1,g1,0)`, line 4, states that in cell `c1`, the gene `g1` is expressed with a value `0`. In line 8, a rule defines a predicate `pert/4` (of arity 4 and composed of 4 terms), modeling the experimental data, in which gene G is expressed at a value S in cell C, which belongs to class CL. In line 9, the program selects a subset of predicates `pert/4` using the predicates `sel_pert/4`, representing the selected perturbations. This type of construct is called *choice rules* and is central in ASP modeling since it generates the possible combinations of candidate solutions. Candidate solutions are usually then filtered using constraints. For example, in line 10, the program forbids the solution candidate `sel_pert/4` associated with the class *early_TE*. Finally, in line 11, the program shows the answer to this program, focusing only on the predicate `sel_pert/4`. This answer is given by a set of answers (answer sets) of assignments of constants to the terms of predicate `sel_pert/4`; each assignment verifies all program rules.

In conclusion, ASP provides a powerful way of specifying complex rules and constraints, making it a valuable tool for solving a wide range of problems in areas such as artificial intelligence, natural language processing, and planning.

2 Pseudo-perturbations maximization program

In this second section, we explain line by line the program, implemented in ASP, used to maximize the number of pseudo-perturbations, which we expose the algorithm in our paper (see Section 3.3, Maximizing the number of pseudo-perturbations). First, this program is based on the method proposed in [2], but mainly differs in the rule for generating different Boolean pseudo-perturbation vectors imposed in our program. Our logic program is specific to scRNAseq data, which is expected to be redundant as some cells in the same developmental stage have the same gene expression. Another specificity of scRNAseq data is the strong abundance of zero values.

In Listing 2, we present the ASP encoding.

```
1  {selgene(G):pert(C,G,S,CL)} = k.
2  selpert(C,G,S,CL) :- selgene(G), pert(C,G,S,CL).
3  equal(I,J,G) :- selpert(I,G,S1,C1), selpert(J,G,S2,C2), C1<C2, S1 = S2.
4  countequal(I,J,M) :- M={equal(I,J,_)}.
5  0{affinity(I,J)}1 :- countequal(I,J,k).
6  nbInputOnes(C, N) :- N={pert(C,G,1,_) : selinput(G)}, affinity(C,_).
7  :- affinity(C,_), nbInputOnes(C,N), N<1.
8  diff(I1,I2,G) :- selpert(I1,G,S1,C1), selpert(I2,G,S2,C2), C1==C2, S1!=S2,
       I1<I2.
9  countdiff(I1,I2,M) :- M={diff(I1,I2,_)}.
10 :- countdiff(I1,I2,0), affinity(I1,_), affinity(I2,_), I1<I2.
11 :- countdiff(I1,I2,0), affinity(_,I1), affinity(_,I2), I1<I2.
12 #maximize{1,I: affinity(I,_)}
```

Listing 2: ASP encoding of pseudo-perturbation generation

Predicate `pert/4` is the instance of our program, referring to the experimental data. It is derived from the discretization of scRNAseq data related to input and intermediate genes. It states that gene G is expressed at a value S in cell C, which belongs to class CL. Our logic program begins (line 1) with selecting a set of k genes from all possible (input and intermediate) genes using the `selgene/1` predicate. This step generates $\binom{m}{k}$ answer sets, where m is the total number of input and intermediate genes. The following rules are meant to filter these candidate answer sets. In line 2, the `selpert/4` predicate summarizes the experimental data for the selected genes. Then, in line 3, using the `equal(I,J,G)` predicate, we select pairs of cells I and J that belong to different classes (`C1<C2`), in which gene G is measured with the same value S (`S1=S2`). In line 4, the `countequal(I,J,M)` predicate counts the number of genes M for which their values are equal across cells I and J. Recall that we are interested in finding k identical values associations for k genes. Therefore, in line 5, we define the predicate `affinity(I,J)`, which will be generated 0 or 1 times when there are k similarities for cells I and J. This predicate will identify the candidate optimal pseudo-perturbations. Note that the left and right terms of predicate `affinity/2`, defined in this case by variables I and J, refer to cells in the first, respectively, second class.

To address data sparsity, we added the rules in lines 6-7. Line 6 defines `nbInputOnes/2`, which counts the number of input genes whose value is `1` for a cell C that has been selected by the predicate `affinity/2`. Then, in line 7, we forbid selecting an `affinity(C,_)` if the number of 1-signed input genes in cell C is less than 1 (`N<1`). These rules ensure that at least one input gene is expressed to 1 for each pseudo-perturbation. This also implies that vectors with all genes inactive (equal to 0) are not allowed.

To identify distinct pseudo-perturbations, we introduced new rules (lines 8-11). Line 8 defines `diff(I1,I2,G)` predicate, which selects cells $I1$ and $I2$, from the same class, with different values for gene G. Then, in line 9, a predicate `countdiff/3` stores the differences in the selected genes' expression values for cells $I1$ and $I2$. In line 10 (resp. line 11), the constraint forbids predicates `countdiff(I1,I2,0)`, where there is no difference in expression values for the selected genes, for cells $I1$ and $I2$ selected to be affinities in line 5 for the first class (resp. for the second class). Combined lines 5, 10, and 11 keep only one cell association when the same cell is associated with other cells, such as retaining only `affinity(c1,c2)` and not `affinity(c1,c3)` from possible associations.

Finally, in line 12, we search to maximize the associations given by predicate `affinity/2` concerning the first class, left term.

After obtaining the optimal pseudo-perturbations, we use Python to find the readout values, which maximize the expression difference in both classes. Altogether, the optimal pseudo-perturbations associated with readouts having a maximal difference between classes constitute the experimental design for cells in classes A and B which will be the input to Caspo for inferring BNs specific to each class.

To sum up, our approach involves augmenting the initial logic program proposed by Chebouba *et al.* with various constraints and rules, thereby imposing further restrictions on the problem and facilitating its solution for the solvers. Note that our proposed approach addresses a more constrained problem, removing redundant pseudo-perturbations retained in Chebouba *et al.*'s version.

References

[1] Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press, New York, NY, USA (2003)

[2] Chebouba, L., Miannay, B., Boughaci, D., Guziolowski, C.: Discriminate the response of acute myeloid leukemia patients to treatment by using proteomics data and answer set programming. BMC Bioinformatics **19**(2), 15–26 (2018)