

A constraint programming implementation for A logical approach to context-specific independence

Jukka Corander^{a,c}, Antti Hyttinen^b, Juha Kontinen^a, Johan Pensar^d, Jouko Väänänen^{a,e}

^a*Department of Mathematics and Statistics, University of Helsinki*

^b*HIIT, Department of Computer Science, University of Helsinki*

^c*Department of Biostatistics, University of Oslo*

^d*Department of Mathematics and Statistics, Åbo Akademi University*

^e*Institute for Logic, Language and Computation, University of Amsterdam*

1. Introduction

We present here the implementation using answer set programming (ASP) as the constraint satisfaction formalism[1, 2, 3]. It offers an expressive declarative modeling language, in terms of first-order logical rules, for various types of NP-hard search and optimization problems. To solve a problem via ASP, one first needs to develop an ASP program (in terms of ASP rules/constraints) that models the problem at hand; that is, the declarative rules implicitly represent the solution. Then a solution can be obtained by invoking an off-the-shelf ASP solver, such as the state-of-the-art **Clingo** system [3]. The search algorithms implemented in the **Clingo** system are extensions of state-of-the-art Boolean satisfiability techniques which can today outperform even specialized domain-specific algorithms.

We present here a code that takes an LDAG as input and outputs all independence relations that the axiomatization is able to derive. The aim is not to provide the most efficient solution but to give concrete implementation which can be used for various purposes, such as examining the conjectured

Email addresses: `jukka.corander@helsinki.fi` (Jukka Corander),
`antti.hyttinen@helsinki.fi` (Antti Hyttinen), `juha.kontinen@helsinki.fi` (Juha Kontinen),
`jopensar@abo.fi` (Johan Pensar), `jouko.vaananen@helsinki.fi` (Jouko Väänänen)

completeness of the system. Due to representations of contexts the code assumes binary variables, but generalization should be straightforward.

2. Input

The input consists of local Markov conditions for the LDAG. The input is represented using Figure 5 as an example.

```
nodes(4).
inputind(2**(1-1),0,2**(2-1) ? 2**(3-1)). % 1 _||_ 0 |2,3
inputind(2**(2-1),2**(3-1),0). % 2 _||_3
inputind(2**(3-1),2**(2-1),0). % 3 _||_2
inputind(2**(4-1),2**(2-1),2**(1-1) ? 2**(3-1)). % 4 _||_ 2 | 1,3
```

Here the predicate `inputind(X,Y,Z)` marks the fact that set X is independent from set Y when conditioning on S . The notation `**` stands for exponentiation. The sets are defined as the integers where bits are on if the element is in the set. In the notation `?` stands for bitwise OR operation. The predicate here must be different from the `ind` (output), otherwise the proof of Theorem 27 would need modifications. Predicate `nodes` defines the number of nodes. Predicate `positive` should be defined if positivity is assumed, this turns on the intersection axiom.

In addition we need to define the labels. We directly define the set $A(e_c)$.

```
% Graph under context 3=0 arises by removing 2->1
contextind(2**(1-1),2**(2-1),2**(3-1),2**(3-1),0).
% Graph under context 3=1 arises by removing 1->4
contextind(2**(4-1),2**(1-1),2**(3-1),2**(3-1),2**(3-1)).
```

3. Encoding

The following encoding can be found in `ldag_csi_paper.pl`.

3.1. Preliminary definitions

First, some preliminary definitions are needed.

```
% Definition of a set as all subsets of 1..N.
set(0..((2**N)-1)) :-nodes(N).

% This predicate says that context SE is valid for S.
% context Se is defined as bit SE[i] being 0 or 1.
% SE[i] is assumed 0 if i does not appear in context S.
context(C,EC) :- set(C), set(EC), C ? EC == C.

disjoint(A,B) :- set(A),set(B),A & B = 0.
```

In the notation ? stands for bitwise OR operation, & stands for bitwise AND operation.

3.2. Taking Input Independencies to the Contexts

The input independencies are directly taken to the context specific derivations.

```
contextind(X,Y,Z,C,EC) :- inputind(X,Y,Z),context(C,EC).
```

3.3. Semigraphoid Axioms in the Contexts

These rules implement the semigraphoid axioms in the context specific derivations, following Definition 22. The preconditions are used to force sets disjoint.

```
% triviality
contextind(X,0,Z,C,EC) :- disjoint(X,Z), context(C,EC).

% symmetry
contextind(Y,X,Z,C,EC) :- contextind(X,Y,Z,C,EC).
```

```

% decomposition
contextind(X,Y,Z,C,EC) :- contextind(X,Y ? U,Z,C,EC),
                           disjoint(Y,U).

% weak union
contextind(X,Y,Z ? U,C,EC) :- contextind(X,Y ? U,Z,C,EC),
                              disjoint(Y,U).

% contraction
contextind(X,Y ? U,Z,C,EC) :- contextind(X,Y,Z ? U,C,EC),
                              contextind(X,U,Z,C,EC).

% intersection (only if positivity is assumed)
contextind(X,Y ? Z,U,C,EC) :- contextind(X,Y,Z ? U,C,EC),
                              contextind(X,Z,Y ? U,C,EC),
                              disjoint(Z,U),disjoint(Y,U),
                              positive.

```

Note that these independences could be deduced using d-separation instead, see [4] for an example encoding. Note the difference of the predicates: **ind** denotes a valid CSI, while **contextind** denotes d-separation relation in the context specific DAG.

3.4. The CSI-rule

The CSI-rule takes independencies from the context specific derivations to the global level.

```

ind(A,B,S,C,CE) :- contextind(A,B,C ? S,C,CE), disjoint(C,S).

```

3.5. The RC-rule

The RC-rules generalize and specialize contexts.

```

ind(A,B,C2 ? S,C1,EC1) :- ind(A,B,S,C,EC), %the first fixes A,B etc.
                           set(C2), C2 ? C = C, C2 != 0,
                           %C2 is a nonempty subset of C (eq. allowed)
                           C1 = C-C2, EC1 = EC & C1,
                           %C1 is the remaining, fixing also EC1
                           ind(A,B,S,C,ECC): context(C2,EC2), ECC=EC1 ? EC2.
                           %we have to have ind for all these

ind(A,B,S,C ? W,EC ? EW) :- disjoint(C,S), context(C,EC), C != 0,
                           ind(A,B,C ? S,W,EW).

```

3.6. Semi-graphoid Axioms on the Global Level

Semi-graphoid axioms can also be used at the global level, following p. 17.

```

% triviality
ind(A,0,S,C,EC) :- disjoint(A,S),disjoint(A,C),
                   disjoint(S,C),context(C,EC).

% symmetry
ind(B,A,S,C,EC) :- ind(A,B,S,C,EC).

% decomposition
ind(A,B,S,C,EC) :- ind(A,B ? BB,S,C,EC), disjoint(B,BB).

% weak union
ind(A,B,S ? BB,C,EC) :- ind(A,B ? BB,S,C,EC), disjoint(B,BB).

% contraction
ind(A,B ? W,S,C,EC) :- ind(A,B,S ? W,C,EC), ind(A,W,S,C,EC).

% intersection (only if positivity is defined)
ind(A,B ? W,S,C,EC) :- ind(A,B,S ? W,C,EC),
                       ind(A,W,S ? B,C,EC),
                       disjoint(S,W),disjoint(S,B),
                       positive.

```

3.7. Output Format

The following shows the global independencies that were derived.

```
#show.  
#show ind/5.
```

In addition the perl script `output.pl` takes the output of `Clingo` and takes out the trivial independencies and switches to more readable set format.

4. Example in Figure 5

When the input in Section 2 is stored in `fig5.pl`, the command:

```
./clingo ldag_csi_paper.pl fig5.pl | ./output.pl
```

gives output:

```
1 _||_ 2 | 3=0  
2 _||_ 3  
1 _||_ 4 | 3=1  
2 _||_ 4 | 1,3  
2 _||_ 4 | 3, 1=0  
2 _||_ 4 | 1, 3=0  
2 _||_ 4 | 1=0,3=0  
2 _||_ 4 | 3, 1=1  
2 _||_ 4 | 1=1,3=0  
2 _||_ 4 | 1, 3=1  
2 _||_ 4 | 1=0,3=1  
2 _||_ 4 | 1=1,3=1  
2 _||_ 1,4 | 3=0  
1,2 _||_ 4 | 3=1  
2 _||_ 4 | 3=0  
1 _||_ 4 | 2, 3=1  
2 _||_ 4 | 3=1
```

```

1 _||_ 2 | 4, 3=0
1 _||_ 4 | 2=0,3=1
1 _||_ 4 | 2=1,3=1
1 _||_ 2 | 3=0,4=0
1 _||_ 2 | 3=0,4=1
2 _||_ 4 | 3
2 _||_ 3,4
2 _||_ 3 | 4
2 _||_ 4
2 _||_ 3 | 4=0
2 _||_ 3 | 4=1

```

Thus, the important independence $2 _||_ 4$ is derived.

5. Example in Figure 6

The command:

```
./clingo ldag_csi_paper.pl fig6.pl | ./output.pl
```

gives output:

```

...
1 _||_ 2 | 3=0,4=0
1 _||_ 2 | 3=0,4=1
1 _||_ 2,5 | 3=0,4=0
1 _||_ 2,5 | 3=0,4=1
1 _||_ 2 | 5, 3=0,4=0
1 _||_ 2 | 4, 3=0,5=0
1 _||_ 2 | 3=0,4=0,5=0
1 _||_ 2 | 5, 3=0,4=1
1 _||_ 2 | 3=0,4=1,5=0
1 _||_ 2 | 4, 3=0,5=1
1 _||_ 2 | 3=0,4=0,5=1
1 _||_ 2 | 3=0,4=1,5=1
1 _||_ 5 | 3=0,4=0

```

```

1 _||_ 5 | 3=0,4=1
4 _||_ 2,5 | 3
4 _||_ 5 | 3
2 _||_ 4 | 3
2 _||_ 4 | 3,5
2 _||_ 4 | 3, 5=0
2 _||_ 4 | 3, 5=1
2 _||_ 3,4 | 5=1
2 _||_ 3 | 4, 5=1
2 _||_ 4 | 5=1
2 _||_ 3 | 4=0,5=1
2 _||_ 3 | 4=1,5=1

```

Thus, the important independence $2 _||_ 4 \mid 5=1$ is derived.

References

- [1] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, *Annals of Mathematics and Artificial Intelligence* 25 (3-4) (1999) 241–273.
- [2] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, *Artificial Intelligence* 138 (1-2) (2002) 181–234.
- [3] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. Schneider, Potassco: The Potsdam answer set solving collection, *AI Communications* 24 (2) (2011) 107–124.
- [4] A. Hyttinen, F. Eberhardt, M. Järvisalo, Constraint-based causal discovery: Conflict resolution with answer set programming, in: *Proc. UAI*, AUAI Press, 2014, pp. 340–349.