Generating Random Logic Programs Using Constraint Programming

Paulius Dilkas[1] Vaishak Belle[1,2]

[1]University of Edinburgh, Edinburgh, UK

[2]Alan Turing Institute, London, UK

CP 2020

THE UNIVERSITY of EDINBURGH informatics

EDINBURGH CENTRE FOR ROBOTICS

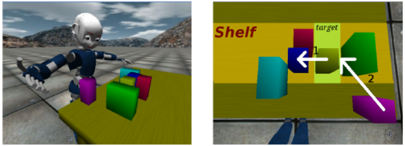
UKRI Engineering and Physical Sciences Research Council

Probabilistic Logic Programs (ProbLog)

The Smokers Network (Domingos et al. 2008)

```
0.2::stress(P) :- person(P).
0.3::influences(P1, P2) :- friend(P1, P2).
0.1::cancer_spont(P) :- person(P).
0.3::cancer_smoke(P) :- person(P).
     smokes(X) :- stress(X).
     smokes(X) :- smokes(Y), influences(Y, X).
     cancer(P) :- cancer_spont(P).
     cancer(P) :- smokes(P), cancer_smoke(P).
     person(michelle).
     person(timothy).
     friend(timothy, michelle).
```

Applications



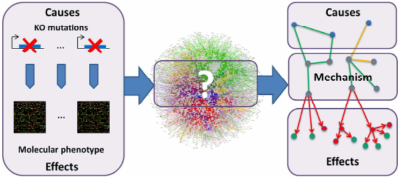
Moldovan et al. 2012

Q1: In a group of 10 people, 60 percent have brown eyes. Two people are to be selected at random from the group. What is the probability that neither person selected will have brown eyes?
Q2: Mike has a bag of marbles with 4 white, 8 blue, and 6 red marbles. He pulls out one marble from the bag and it is red. What is the probability that the second marble he pulls out of the bag is white?

Dries et al. 2017

```
is_malignant(Case):-
        biopsyProcedure(Case,usCore),
        changes_Sizeinc(Case,missing),
        feature_shape(Case).
is_malignant(Case):-
        assoFinding(Case,asymmetry),
        breastDensity(Case,scatteredFDensities),
        vacuumAssisted(Case,yes).
is_malignant(Case):-
        needleGauge(Case,9),
        offset(Case,14),
        vacuumAssisted(Case,yes).
```

Côrte-Real, Dutra, and Rocha 2017



De Maeyer et al. 2013

Inference Algorithms and Knowledge Compilation Maps

NNF	negation normal form
d-DNNF	deterministic decomposable negation normal form
BDD	binary decision diagrams
SDD	sentential decision diagrams
k-Best	???

Example Diagrams for $C \wedge (A \vee \neg B)$

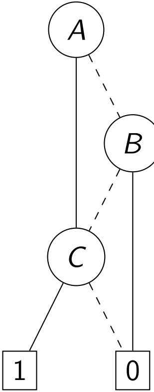


Figure: BDD

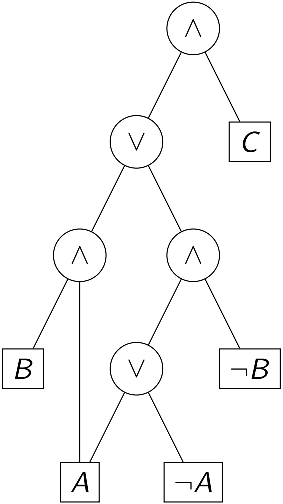


Figure: d-DNNF

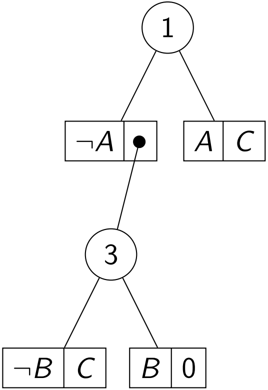


Figure: SDD

How Many Programs Are Used to Test Algorithms?

How Many Programs Are Used to Test Algorithms?

Anytime Inference in Probabilistic Logic Programs with $T_{\mathcal{P}}$-Compilation

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, Luc De Raedt
Department of Computer Science
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

4

How Many Programs Are Used to Test Algorithms?

Anytime Inference in Probabilistic Logic Programs with $T_{\mathcal{P}}$-Compilation

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, Luc De Raedt
Department of Computer Science
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

4

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS and LUC DE RAEDT
Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(*e-mail:* FirstName.LastName@cs.kuleuven.be)

3

How Many Programs Are Used to Test Algorithms?

Anytime Inference in Probabilistic Logic Programs with $T_{\mathcal{P}}$-Compilation

4

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, Luc De Raedt
Department of Computer Science
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

Inference and learning in probabilistic logic programs using weighted Boolean formulas

3

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS, DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON, GERDA JANSSENS and LUC DE RAEDT
Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(*e-mail:* FirstName.LastName@cs.kuleuven.be)

***k*-Optimal: a novel approximate inference algorithm for ProbLog**

1

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen

How Many Programs Are Used to Test Algorithms?

Anytime Inference in Probabilistic Logic Programs with $T_\mathcal{P}$-Compilation

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, Luc De Raedt
Department of Computer Science
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

4

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS, DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON, GERDA JANSSENS and LUC DE RAEDT
Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(*e-mail:* FirstName.LastName@cs.kuleuven.be)

3

***k*-Optimal: a novel approximate inference algorithm for ProbLog**

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen

1

On the Efficient Execution of ProbLog Programs

Angelika Kimmig[1], Vítor Santos Costa[2], Ricardo Rocha[2], Bart Demoen[1], and Luc De Raedt[1]

1

How Many Programs Are Used to Test Algorithms?

Anytime Inference in Probabilistic Logic Programs with T_P-Compilation

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, Luc De Raedt
Department of Computer Science
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

4

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS, DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON, GERDA JANSSENS and LUC DE RAEDT
Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(*e-mail:* FirstName.LastName@cs.kuleuven.be)

3

***k*-Optimal: a novel approximate inference algorithm for ProbLog**

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen

1

On the Efficient Execution of ProbLog Programs

Angelika Kimmig[1], Vítor Santos Costa[2], Ricardo Rocha[2], Bart Demoen[1], and Luc De Raedt[1]

1

On the Implementation of the Probabilistic Logic Programming Language ProbLog

Angelika Kimmig, Bart Demoen and Luc De Raedt
Departement Computerwetenschappen, K.U. Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
(*e-mail:* {Angelika.Kimmig,Bart.Demoen,Luc.DeRaedt}@cs.kuleuven.be)

Vítor Santos Costa and Ricardo Rocha
CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal
(*e-mail:* {vsc,ricroc}@dcc.fc.up.pt)

1

How Many Programs Are Used to Test Algorithms?

Anytime Inference in Probabilistic Logic Programs with T_P-Compilation

4

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, Luc De Raedt
Department of Computer Science
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

Inference and learning in probabilistic logic programs using weighted Boolean formulas

3

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS, DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON, GERDA JANSSENS and LUC DE RAEDT
Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(*e-mail:* FirstName.LastName@cs.kuleuven.be)

***k*-Optimal: a novel approximate inference algorithm for ProbLog**

1

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen

On the Efficient Execution of ProbLog Programs

1

Angelika Kimmig[1], Vítor Santos Costa[2], Ricardo Rocha[2], Bart Demoen[1], and Luc De Raedt[1]

On the Implementation of the Probabilistic Logic Programming Language ProbLog

1

Angelika Kimmig, Bart Demoen and Luc De Raedt
Departement Computerwetenschappen, K.U. Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
(*e-mail:* {Angelika.Kimmig,Bart.Demoen,Luc.DeRaedt}@cs.kuleuven.be)

Vítor Santos Costa and Ricardo Rocha
CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal
(*e-mail:* {vsc,ricroc}@dcc.fc.up.pt)

ProbLog Technology for Inference in a Probabilistic First Order Logic

2

Maurice Bruynooghe and **Theofrastos Mantadelis** and **Angelika Kimmig** and **Bernd Gutmann** and **Joost Vennekens** and **Gerda Janssens** and **Luc De Raedt**[1]

What Characterises a (Probabilistic) Logic Program?

$0.2 :: \texttt{stress}(\mathsf{P}) \texttt{:-} \texttt{person}(\mathsf{P}).$

$0.3 :: \texttt{influences}(\mathsf{P}_1, \mathsf{P}_2) \texttt{:-} \texttt{friend}(\mathsf{P}_1, \mathsf{P}_2).$

$0.1 :: \texttt{cancer\_spont}(\mathsf{P}) \texttt{:-} \texttt{person}(\mathsf{P}).$

$0.3 :: \texttt{cancer\_smoke}(\mathsf{P}) \texttt{:-} \texttt{person}(\mathsf{P}).$

$\texttt{smokes}(\mathsf{X}) \texttt{:-} \texttt{stress}(\mathsf{X}).$

$\texttt{smokes}(\mathsf{X}) \texttt{:-} \texttt{smokes}(\mathsf{Y}), \texttt{influences}(\mathsf{Y}, \mathsf{X}).$

$\texttt{cancer}(\mathsf{P}) \texttt{:-} \texttt{cancer\_spont}(\mathsf{P}).$

$\texttt{cancer}(\mathsf{P}) \texttt{:-} \texttt{smokes}(\mathsf{P}), \texttt{cancer\_smoke}(\mathsf{P}).$

$\texttt{person}(\mathit{michelle}).$

$\texttt{person}(\mathit{timothy}).$

$\texttt{friend}(\mathit{timothy}, \mathit{michelle}).$

What Characterises a (Probabilistic) Logic Program?

$0.2 :: \texttt{stress}(\mathsf{P}) \;\text{:-}\; \texttt{person}(\mathsf{P}).$

$0.3 :: \texttt{influences}(\mathsf{P}_1, \mathsf{P}_2) \;\text{:-}\; \texttt{friend}(\mathsf{P}_1, \mathsf{P}_2).$

$0.1 :: \texttt{cancer\_spont}(\mathsf{P}) \;\text{:-}\; \texttt{person}(\mathsf{P}).$

$0.3 :: \texttt{cancer\_smoke}(\mathsf{P}) \;\text{:-}\; \texttt{person}(\mathsf{P}).$

$\texttt{smokes}(\mathsf{X}) \;\text{:-}\; \texttt{stress}(\mathsf{X}).$

$\texttt{smokes}(\mathsf{X}) \;\text{:-}\; \texttt{smokes}(\mathsf{Y}), \texttt{influences}(\mathsf{Y}, \mathsf{X}).$

$\texttt{cancer}(\mathsf{P}) \;\text{:-}\; \texttt{cancer\_spont}(\mathsf{P}).$

$\texttt{cancer}(\mathsf{P}) \;\text{:-}\; \texttt{smokes}(\mathsf{P}), \texttt{cancer\_smoke}(\mathsf{P}).$

$\texttt{person}(\mathit{michelle}).$

$\texttt{person}(\mathit{timothy}).$

$\texttt{friend}(\mathit{timothy}, \mathit{michelle}).$

- predicates, arities

What Characterises a (Probabilistic) Logic Program?

$0.2 :: \texttt{stress}(P) \mathrel{:\!-} \texttt{person}(P).$

$0.3 :: \texttt{influences}(P_1, P_2) \mathrel{:\!-} \texttt{friend}(P_1, P_2).$

$0.1 :: \texttt{cancer\_spont}(P) \mathrel{:\!-} \texttt{person}(P).$

$0.3 :: \texttt{cancer\_smoke}(P) \mathrel{:\!-} \texttt{person}(P).$

$\texttt{smokes}(X) \mathrel{:\!-} \texttt{stress}(X).$

$\texttt{smokes}(X) \mathrel{:\!-} \texttt{smokes}(Y), \texttt{influences}(Y, X).$

$\texttt{cancer}(P) \mathrel{:\!-} \texttt{cancer\_spont}(P).$

$\texttt{cancer}(P) \mathrel{:\!-} \texttt{smokes}(P), \texttt{cancer\_smoke}(P).$

$\texttt{person}(\mathit{michelle}).$

$\texttt{person}(\mathit{timothy}).$

$\texttt{friend}(\mathit{timothy}, \mathit{michelle}).$

- predicates, arities
- variables

What Characterises a (Probabilistic) Logic Program?

```
0.2::stress(P) :- person(P).
0.3::influences(P1, P2) :- friend(P1, P2).
0.1::cancer_spont(P) :- person(P).
0.3::cancer_smoke(P) :- person(P).
     smokes(X) :- stress(X).
     smokes(X) :- smokes(Y), influences(Y, X).
     cancer(P) :- cancer_spont(P).
     cancer(P) :- smokes(P), cancer_smoke(P).
     person(michelle).
     person(timothy).
     friend(timothy, michelle).
```

- predicates, arities
- variables
- constants

What Characterises a (Probabilistic) Logic Program?

```
0.2::stress(P) :- person(P).
0.3::influences(P1, P2) :- friend(P1, P2).
0.1::cancer_spont(P) :- person(P).
0.3::cancer_smoke(P) :- person(P).
     smokes(X) :- stress(X).
     smokes(X) :- smokes(Y), influences(Y, X).
     cancer(P) :- cancer_spont(P).
     cancer(P) :- smokes(P), cancer_smoke(P).
     person(michelle).
     person(timothy).
     friend(timothy, michelle).
```

- predicates, arities
- variables
- constants
- probabilities

What Characterises a (Probabilistic) Logic Program?

```
0.2::stress(P) :- person(P).
0.3::influences(P1, P2) :- friend(P1, P2).
0.1::cancer_spont(P) :- person(P).
0.3::cancer_smoke(P) :- person(P).
     smokes(X) :- stress(X).
     smokes(X) :- smokes(Y), influences(Y, X).
     cancer(P) :- cancer_spont(P).
     cancer(P) :- smokes(P), cancer_smoke(P).
     person(michelle).
     person(timothy).
     friend(timothy, michelle).
```

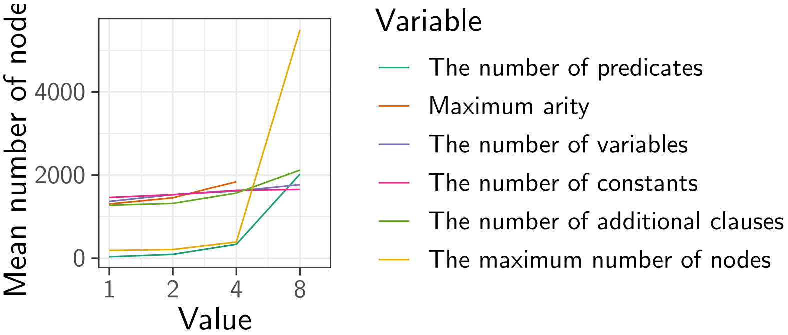
- predicates, arities
- variables
- constants
- probabilities
- length

What Characterises a (Probabilistic) Logic Program?

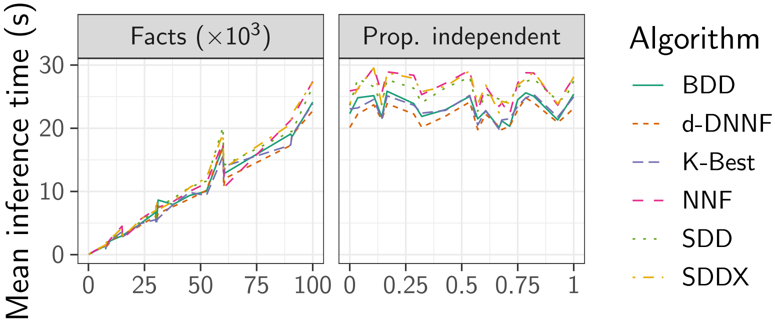
$$
\left\{
\begin{array}{l}
0.2 :: \texttt{stress}(P) :\text{-}\ \texttt{person}(P).\\
0.3 :: \texttt{influences}(P_1, P_2) :\text{-}\ \texttt{friend}(P_1, P_2).\\
0.1 :: \texttt{cancer\_spont}(P) :\text{-}\ \texttt{person}(P).\\
0.3 :: \texttt{cancer\_smoke}(P) :\text{-}\ \texttt{person}(P).\\
\qquad \texttt{smokes}(X) :\text{-}\ \texttt{stress}(X).\\
\qquad \texttt{smokes}(X) :\text{-}\ \texttt{smokes}(Y), \texttt{influences}(Y, X).\\
\qquad \texttt{cancer}(P) :\text{-}\ \texttt{cancer\_spont}(P).\\
\qquad \texttt{cancer}(P) :\text{-}\ \underbrace{\texttt{smokes}(P), \texttt{cancer\_smoke}(P)}.\\
\qquad \texttt{person}(\mathit{michelle}).\\
\qquad \texttt{person}(\mathit{timothy}).\\
\qquad \texttt{friend}(\mathit{timothy}, \mathit{michelle}).
\end{array}
\right.
$$

- predicates, arities
- variables
- constants
- probabilities
- length
- complexity

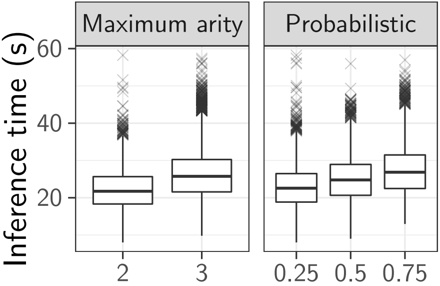
Scalability



Properties of Programs vs. Inference Algorithms



Properties of Programs vs. Inference Algorithms



Overview

General parameters

- maximum number of solutions
- `maxNumNodes` (in the tree representation of a clause)
- list of predicates with their variables
- maximum number of clauses
- option to forbid all cycles or just negative cycles
- list of probabilities that are randomly assigned to clauses: $\{0.1, 0.2, \dots, 0.9, 1, 1, 1, 1, 1, 1\}$

Decision variables

- `IntVar[] clauseAssignments`: a predicate or disabled
- `Clause[] clauses`

Constraints

Each predicate should get at least one constraint

- `numDisabledClauses`: defined by a count constraint
- $\texttt{numDistinctValues} = \begin{cases} \texttt{numPredicates} + 1 & \text{if } \texttt{numDisabledValues} > 0 \\ \texttt{numPredicates} & \text{otherwise.} \end{cases}$
  - also constrained using the `nValues` constraint

Miscellaneous

- `clauseAssignments` are sorted.
- If $\texttt{clauseAssignments}[i-1] = \texttt{clauseAssignments}[i]$,
  - then $\texttt{clause}[i-1] \preceq \texttt{clause}[i]$.

Clauses

A clause is defined by...

- `IntVar[] treeStructure`
  - $\texttt{treeStructure}[i] = i$: the i-th node is a root.
  - $\texttt{treeStructure}[i] = j$: the i-th node's parent is node j.
- `IntVar[] treeValues`: $\neg$, $\wedge$, $\vee$, $\top$, and any predefined predicates with variables.

Auxiliary variables

- $\texttt{numNodes}, \texttt{numTrees} \in \{1, \ldots, \texttt{maxNumNodes}\}$

Clause constraints

- $\texttt{treeStructure}$ represents $\texttt{numTrees}$ trees.
- $\texttt{treeStructure}[0] = 0$
- $\texttt{numTrees} + \texttt{numNodes} = \texttt{maxNumNodes} + 1$
- $\texttt{treeStructure}$ is sorted
- For $i = 0, \ldots, \texttt{maxNumNodes} - 1$,
  - If $\texttt{numNodes} \leq i$,
  - then $\texttt{treeStructure}[i] = i$ and $\texttt{treeValues}[i] = \top$,
  - else $\texttt{treeStructure}[i] < \texttt{numNodes}$.
  - has 0 children $\iff$ $\texttt{treeValues}[i]$ is a predicate
  - has 1 child $\iff$ $\texttt{treeValues}[i] = \neg$
  - has > 1 child $\iff$ $\texttt{treeValues}[i] \in \{\wedge, \vee\}$
  - $\texttt{treeStructure}[i] \neq i \implies \texttt{treeValues}[i] \neq \top$
- If the clause should be disabled, $\texttt{numNodes} = 1$ and $\texttt{treeValues}[0] = \top$.

Adjacency matrix representation

$A[i][j] = 0 \iff \nexists k : \texttt{clauseAssignments}[k] = j$ and $i \in \texttt{clauses}[k].\texttt{treeValues}$

New constraints

- No (negative) cycles
  - No clever propagation, just entailment checking.
- Independence. Propagation:
  - Two types of dependencies: determined and one-undetermined-edge-away-from-being-determined.
  - Look up the dependencies of both predicates. For each pair of matching dependencies:
    - If both are determined, fail.
    - If one is determined, the selected edge of the other must not exist.
- Conditional independence
  - Same propagation, but with a ‘filter’ that masks out the expression that the independence is conditioned on.