

# Generating Random Logic Programs Using Constraint Programming

**Paulius Dilkas**<sup>1</sup>    Vaishak Belle<sup>1,2</sup>

<sup>1</sup>University of Edinburgh, Edinburgh, UK

<sup>2</sup>Alan Turing Institute, London, UK

CP 2020



THE UNIVERSITY OF EDINBURGH

**informatics**



EDINBURGH CENTRE FOR

**ROBOTICS**



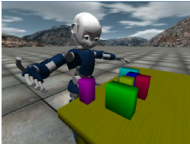
Engineering and  
Physical Sciences  
Research Council

# Probabilistic Logic Programs (PROBLOG)

Smokers (Domingos et al. 2008; Fierens et al. 2015)

```
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

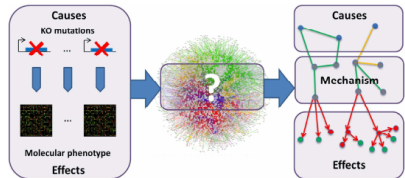
```
is_malignant(Case):-  
    biopsyProcedure(Case,usCore),  
    changes_Sizeinc(Case,missing),  
    feature_shape(Case).  
is_malignant(Case):-  
    assoFinding(Case,asymmetry),  
    breastDensity(Case,scatteredFDensities),  
    vacuumAssisted(Case,yes).  
is_malignant(Case):-  
    needleGauge(Case,9),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

Côte-Real, Dutra, and Rocha 2017

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



De Maeyer et al. 2013

How Many Programs Are Used to Test Algorithms?

# 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

# 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

## *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`)

# 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

## *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)

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

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen

# 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

## On the Efficient Execution of ProbLog Programs

Angelika Kimmig<sup>1</sup>, Vitor Santos Costa<sup>2</sup>, Ricardo Rocha<sup>2</sup>, Bart Demeo<sup>1</sup>, and  
Luc De Raedt<sup>1</sup>

## *Inference and learning in probabilistic logic programs using weighted Boolean formulas*

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,  
DIMITAR SHTERIONOV, BENNO 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**

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen



# 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

## *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)

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

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen

## On the Efficient Execution of ProbLog Programs

Angelika Kimmig<sup>1</sup>, Vítor Santos Costa<sup>2</sup>, Ricardo Rocha<sup>2</sup>, Bart Demoen<sup>1</sup>, and  
Luc De Raedt<sup>1</sup>

## *On the Implementation of the Probabilistic Logic Programming Language ProbLog*

Angelika Kimmig, Bart Demoen and Luc De Raedt  
Department 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 1091/1055, 4169-007 Porto, Portugal  
(e-mail: {vsc,ricroc}@dcc.fc.up.pt)

# 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

## *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)

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

Joris Renkens · Guy Van den Broeck · Siegfried Nijssen

## On the Efficient Execution of ProbLog Programs

Angelika Kimmig<sup>1</sup>, Vítor Santos Costa<sup>2</sup>, Ricardo Rocha<sup>2</sup>, Bart Demoen<sup>1</sup>, and  
Luc De Raedt<sup>1</sup>

## *On the Implementation of the Probabilistic Logic Programming Language ProbLog*

Angelika Kimmig, Bart Demoen and Luc De Raedt  
Department 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 1091/1055, 4169-007 Porto, Portugal  
(e-mail: {vsc,ricroc}@dcc.fc.up.pt)

## ProbLog Technology for Inference in a Probabilistic First Order Logic

Maurice Bruynooghe and Theodoros Mantadelos and Angelika Kimmig and Bernd Gutmann  
and Joost Vennekens and Gerda Janssens and Luc De Raedt<sup>1</sup>

## 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).
```

## What Characterises a (Probabilistic) Logic Program?

0.2::`stress`(P):-`person`(P).

0.3::`influences`(P<sub>1</sub>, P<sub>2</sub>):-`friend`(P<sub>1</sub>, P<sub>2</sub>).

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

## 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

## 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(P<sub>1</sub>,P<sub>2</sub>):-friend(P<sub>1</sub>,P<sub>2</sub>).

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(P<sub>1</sub>,P<sub>2</sub>):-friend(P<sub>1</sub>,P<sub>2</sub>).  
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

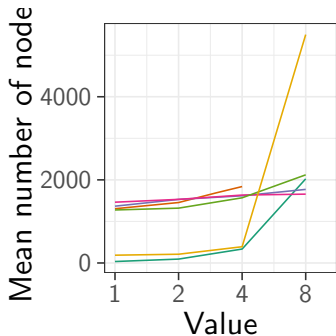


# What Characterises a (Probabilistic) Logic Program?

0.2::stress(P):-person(P).  
0.3::influences(P<sub>1</sub>,P<sub>2</sub>):-friend(P<sub>1</sub>,P<sub>2</sub>).  
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
- complexity

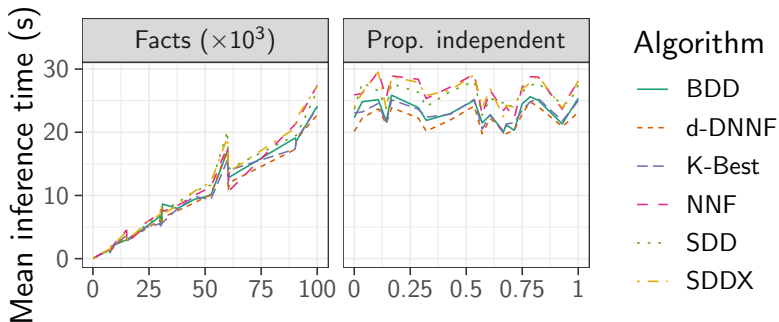
# Scalability



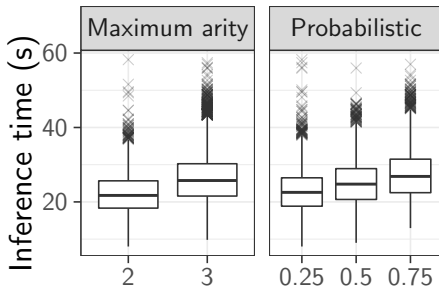
## Variable

- The number of predicates
- Maximum arity
- The number of variables
- The number of constants
- The number of additional clauses
- The maximum number of nodes

# 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
- numDistinctValues =
$$\begin{cases} \text{numPredicates} + 1 & \text{if numDisabledValues} > 0 \\ \text{numPredicates} & \text{otherwise.} \end{cases}$$
  - also constrained using the nValues constraint

## Miscellaneous

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

# Clauses

A clause is defined by...

- `IntVar[] treeStructure`
  - `treeStructure[i] = i`: the  $i$ -th node is a root.
  - `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

- `numNodes, numTrees`  $\in \{1, \dots, \text{maxNumNodes}\}$

## Clause constraints

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



## Adjacency matrix representation

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

## 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.