On the Efficient Execution of ProbLog Programs

Angelika Kimmig, Vítor Santos Costa, Ricardo Rocha, Bart Demoen, Luc De Raedt

Presented by Elad Shtiegmann

Soft-Logic Seminar in Computer Science, 236801
Prof. Benny Kimelfeld

Table of contents

1/5: Presenting ProbLog

Problog is a probabilistic extension to Prolog:

T = { (c,p) }

->

P(query)

1/5: Presenting ProbLog

Problog is a probabilistic extension to Prolog:

T = { (c,p) }

->

P(query)

$$P(L|T) = \prod_{c_i \in L} p_i \prod_{c_i \in L_T \setminus L} (1 - p_i) \quad \Longrightarrow \quad P_s(q|T) = \sum_{L \subseteq L_T} P(L|T)$$

1/5: Presenting ProbLog

ProbLog's inference algorithms

- Exact algorithm X1
- Approximate algorithms X3
  - Bounded Probability, K-best, Iterative Sampling (Monte-Carlo)

1/5: Presenting ProbLog

ProbLog's inference algorithms

- Exact algorithm X1
- Approximate algorithms X3
  - Bounded Probability, K-best, Iterative Sampling (Monte-Carlo)

On top of YAP-Prolog (open source, Universita de Porto)

1/5: Presenting ProbLog - biological networks

A biological network is G(V,E)

V = { biological entities}

E = { biological links }

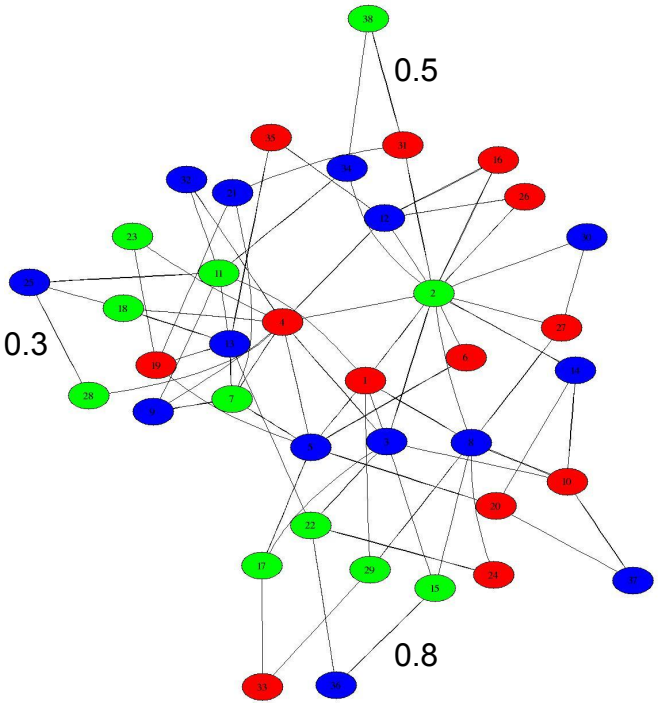
P(e) = P(link between entities)

For machine learning

Probabilistic Logic should be:

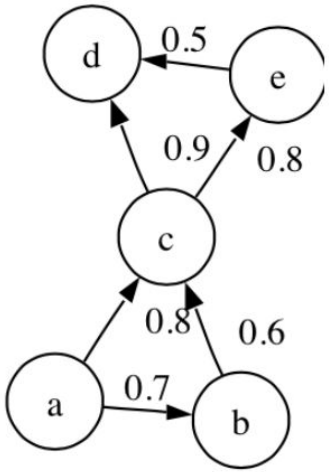
Simple (Learnable)

Efficient (Scalable)



1/5: Presenting ProbLog - example of network

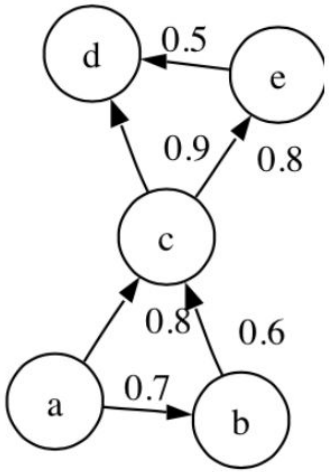
Here is a random graph:



It can be sampled.

1/5: Presenting ProbLog - example of network

Here is a random graph:



It can be sampled.

Here is a ProbLog program:

$$0.8 :: \texttt{edge}(\texttt{a}, \texttt{c}).$$
$$0.6 :: \texttt{edge}(\texttt{b}, \texttt{c}).$$
$$0.7 :: \texttt{edge}(\texttt{a}, \texttt{b}).$$
$$0.9 :: \texttt{edge}(\texttt{c}, \texttt{d}).$$
$$0.8 :: \texttt{edge}(\texttt{c}, \texttt{e}).$$
$$0.5 :: \texttt{edge}(\texttt{e}, \texttt{d}).$$

It can be sampled.

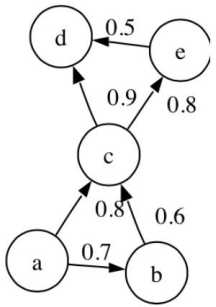
1/5: Presenting ProbLog - ProbLog program

Here is a ProbLog program:

```
0.8 :: edge(a, c).    0.7 :: edge(a, b).    0.8 :: edge(c, e).
0.6 :: edge(b, c).    0.9 :: edge(c, d).    0.5 :: edge(e, d).
```

It can get rules:

```
path(X, Y) :- edge(X, Y).
path(X, Y) :- edge(X, Z), path(Z, Y).
```



Using these rules:

P(path(c,d)) = P(edge(c,d) V (P(edge(c,e) $\wedge$ edge(e,d)))

2/5: Accurate Inference in ProbLog

ProbLog's exact inference algorithms employs two parts:

1. Given a query, find all its proofs
   Result is a DNF formulae (examples ahead)

2. Compute the probability of DNF being satisfied

2/5: Accurate Inference in ProbLog

1. Given a query, find all proofs - Result is a DNF formulae

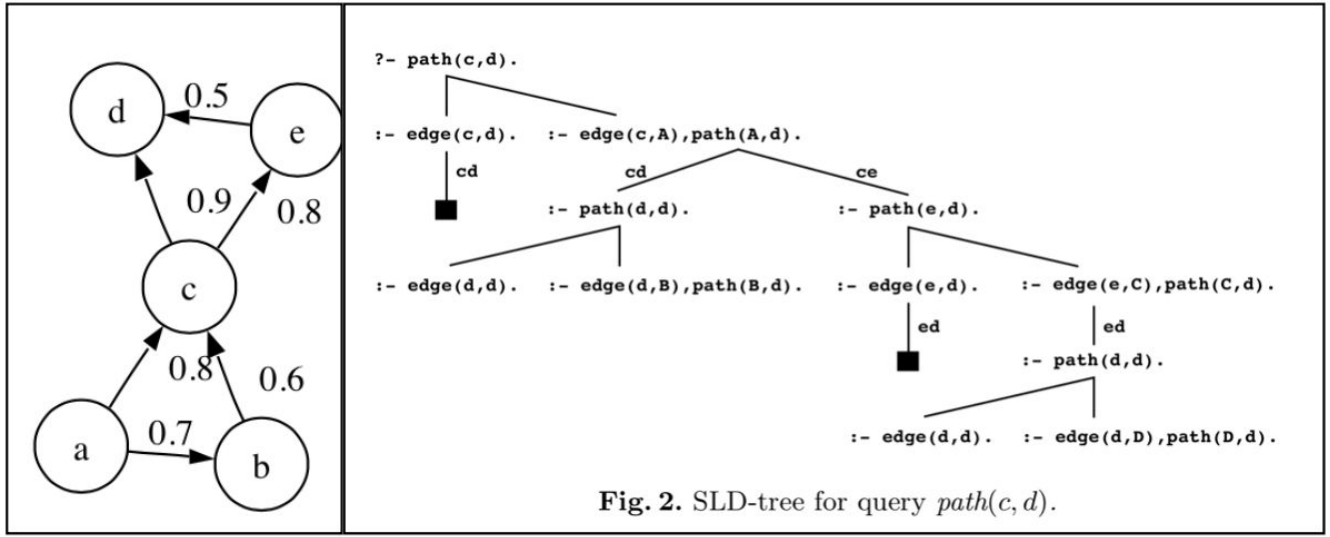


Fig. 2. SLD-tree for query $path(c, d)$.

proof defines set of facts it requires.

2/5: Accurate Inference in ProbLog

1. Given a query, find all proofs - Result is a DNF formulae

   For each proof **p**:

   For each fact **c** in proof:
   Define random variable **b** // P(**b**) = P(**c**)

   Define proof's DNF as b1 ∧ b2 ∧ b3 ∧ .. // P(**q**) = P(**b1**) ∧ P(**b2**) ∧ ...

2/5: Accurate Inference in ProbLog

1. Given a query, find all proofs - Result is a DNF formulae

   Conjunct all DNFs (for all proofs)

$$\bigvee_{e \in E(q)} \bigwedge_{b_i \in cl(e)} b_i$$

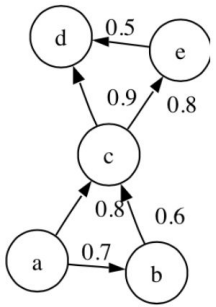
2/5: Accurate Inference in ProbLog

1. Given a query, find all proofs - Result is a DNF formulae

   Conjunct all DNFs (for all proofs)

   Example - path(c,d):

$$cd \vee (ce \wedge ed)$$



   (xy ⇔ edge(x,y))

2/5: Accurate Inference in ProbLog

2. Compute the probability of result DNF being satisfied

$$P_s(q|T) = P\left(\bigvee_{e \in E(q)} \bigwedge_{b_i \in cl(e)} b_i\right)$$

NP-hard

Generally using the exclusion-inclusion principle.
Reasonable for queries with <= 10 proofs

2/5: Approximative Inference in ProbLog

As number of proofs increases the resulting DNF formulae grows

Probability computation becomes infeasible

ProbLog also uses approximate algorithms:

1. Approximate bounding
2. k-best
3. Iterative sampling (Monte-Carlo)

2/5: Approximative Inference in ProbLog

1. Approximate bounding (presented last week)

Produce two DNFs “b” and “t” such that:

$$P(b) \leq P(q) \leq P(t) \qquad P(t) - P(b) < \delta$$

How?

2/5: Approximative Inference in ProbLog

1. Approximate bounding example

while(1)

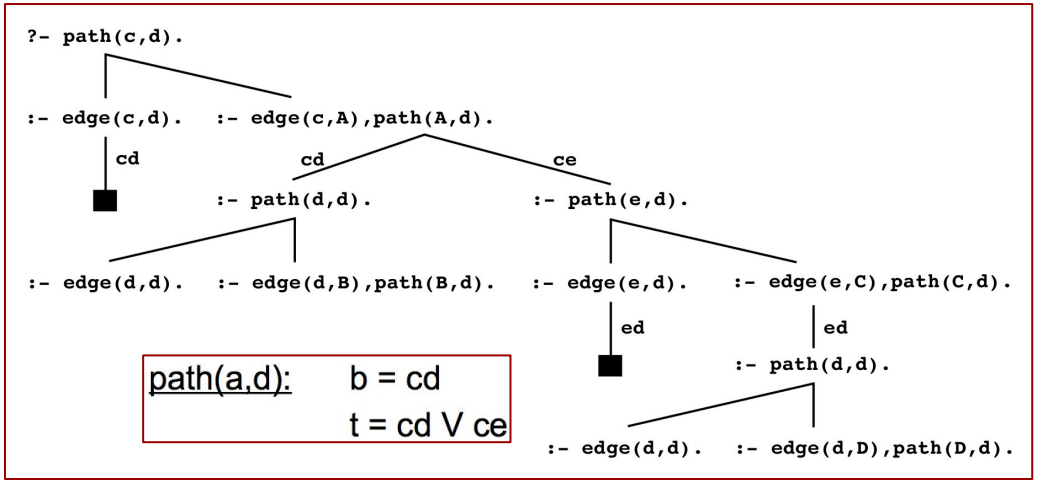
traverse a binary decision tree, extend paths up to some probability threshold

accumulate DNFs

if (difference(b,t) < δ)

return (b,t)

shrink threshold



2/5: Approximative Inference in ProbLog

2. k-best

Produce DNF formulae using only the most probable k proofs

These can be found easily (paper)

1-best corresponds to P(q's most probable proof).

"∞-best" corresponds to P(q).

2/5: Approximative Inference in ProbLog

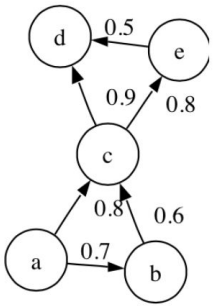
2. K-best example: path(a,d)

P(ac ∧ ce) = 0.72

P(ac ∧ cd) = 0.32

P(ab ∧ bc ∧ cd) = 0.378

P(ab ∧ bc ∧ ce ∧ ed) = 0.168



1-best(q): P (ac ∧ ce) = 0.72

2-best(q): $P((ac \wedge cd) \vee (\neg ac \wedge ab \wedge bc \wedge cd)) = 0.72 + (1-0.8) \cdot 0.378 = 0.7956$

3-best(q): ……. = 0.8276

4-best(q): ……. = 0.83096

2/5: Approximative Inference in ProbLog

3. Iterative Sampling (Monte-Carlo)

do(m times)

generate program sample

check if q exists in sample

re-evaluate probability

calculate δ

// $P\left(P_{approxiamte}(q)-\delta \le P_{exact}(q) \le P_{approxiamte}(q)+\delta\right)=0.95$

repeat until δ is sufficiently small

3/5: ProbLog's Implementation

For scalability, the algorithms suggested require:

1. Preprocessing the database (facts)
   The original ProbLog implementation uses interpreting

2. Data structures to effectively maintain and access:
   Proofs (for bounded approximation and k-best)
   Samples (for iterative sampling):

3/5: ProbLog's Implementation

1. Preprocessing the database

```
0.715 :: edge('PubMed_2196878','MIM_609065').
0.659 :: edge('PubMed_8764571','HGNC_5014').
```

would be compiled as:

```
problog_edge(0,'PubMed_2196878','MIM_609065', −0.3348).
problog_edge(1,'PubMed_8764571','HGNC_5014', −0.4166).
```

maintenance:

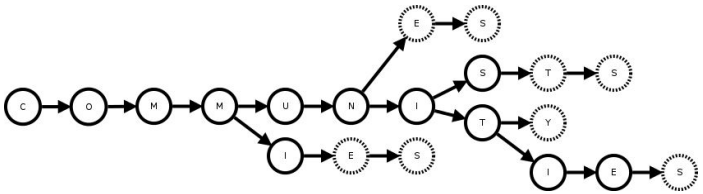
```
edge(A, B) :− problog_edge(C, A, B, D),
              add_to_proof(C, D).
```

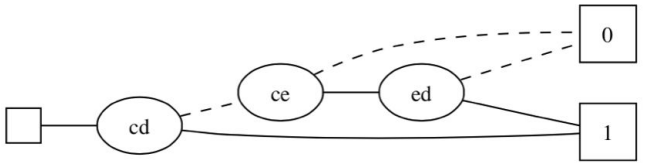
3/5: ProbLog's Implementation

2. Data structures

Proofs (For bounded approximation and k-best):

- Generating DNF
  Proof = list of facts, ordered by first-use
  Some proofs share a prefix
    Use Trie
- Computing DNF's probability
    Use binary trees





$$cd \vee (ce \wedge ed)$$

3/5: ProbLog's Implementation

2. Data structures

Samples (for iterative-sampling):

- Can't generate/store full samples in real programs

  Generate partial samples, expand on-demand (lazy)

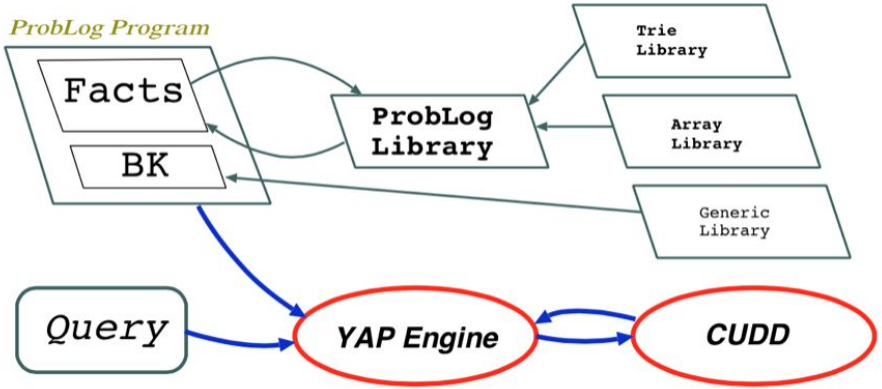
- Sample represented as trinear array of size |facts|:

  s[i] = 0 ⇔ (fact i unsampled)

  s[i] = 1 ⇔ (fact i sampled true)

  s[i] = 2 ⇔ (fact i sampled false)

3/5: ProbLog's Implementation



*minimize trees by bottom-up traversal (linear time)

4/5: Testing our implementation's performance

Test success probabilities of

- k-best
- bounded approximation
- iterative sampling

Using biological networks

- extracted from Biomine's Project
- involving three genes linked to Alzheimer's
  - G582, G620, G983

4/5: Testing our implementation's performance

- Network "SMALL"

  All paths of length <= 2 involving the three gene-nodes

  79 vertices, 144 edges.

- Network "MEDIUM"

  All paths of length <= 3 involving the three gene-nodes

  5220 vertices, 11532 edges.

- Network "BIOMINE"

  All paths

  ~1,000,000 vertices, ~6,000,000 edges

4/5: Testing our implementation's performance

k-best Diagram

path	983 – 620			983 – 582			620 – 582		
k	T_p	T_B	P	T_p	T_B	P	T_p	T_B	P
1	16	-	0.07	4	-	0.03	4	-	0.42
2	0	1613	0.08	0	1686	0.05	4	1511	0.66
4	4	1758	0.10	0	1519	0.06	4	1676	0.86
8	0	1590	0.11	0	1643	0.06	4	1778	0.92
16	4	1744	0.11	4	1536	0.06	4	1719	0.92
32	8	1839	0.11	12	1676	0.07	4	1681	0.96
64	24	1891	0.11	20	1665	0.09	12	1590	0.99
128	52	2054	0.11	32	2130	0.10	48	2286	1.00
256	212	2141	0.11	128	2039	0.10	76	1942	1.00
512	436	13731	0.11	209	2280	0.11	300	2245	1.00
1024	1837	3349	0.11	1372	2195	0.11	581	4080	1.00
exact	641	8343	0.11	5629	2716	0.11	496	2288	1.00

Table 1. k-probability on SMALL.

4/5: Testing our implementation's performance

Network "SMALL"

k-best

path	983 – 620			983 – 582			620 – 582		
k	T_p	T_B	P	T_p	T_B	P	T_p	T_B	P
1	16	-	0.07	4	-	0.03	4	-	0.42
2	0	1613	0.08	0	1686	0.05	4	1511	0.66
4	4	1758	0.10	0	1519	0.06	4	1676	0.86
8	0	1590	0.11	0	1643	0.06	4	1778	0.92
16	4	1744	0.11	4	1536	0.06	4	1719	0.92
32	8	1839	0.11	12	1676	0.07	4	1681	0.96
64	24	1891	0.11	20	1665	0.09	12	1590	0.99
128	52	2054	0.11	32	2130	0.10	48	2286	1.00
256	212	2141	0.11	128	2039	0.10	76	1942	1.00
512	436	13731	0.11	209	2280	0.11	300	2245	1.00
1024	1837	3349	0.11	1372	2195	0.11	581	4080	1.00
exact	641	8343	0.11	5629	2716	0.11	496	2288	1.00

Table 1. k-probability on SMALL.

Faster (than exact algorithm)

4/5: Testing our implementation's performance

Network "SMALL"

bounded approximation

path	$983-620$				$983-582$				$620-582$			
δ	T_p	T_B	n	P	T_p	T_B	n	P	T_p	T_B	n	P
0.1	0	5051	3	[0.07,0.12]	0	4994	3	[0.06,0.12]	12	1690	1	[0.99,1.00]
0.05	0	6504	4	[0.07,0.12]	40	10907	6	[0.06,0.11]	12	1751	1	[0.99,1.00]
0.01	8	9897	6	[0.10,0.11]	68	12684	7	[0.10,0.11]	12	1968	1	[0.99,1.00]
old	T=46234 (x4.5)				T=206300 (x16)				T=307966 (x154)			

* works
* better use accurate algorithm (small problem)
* much better than the original implementation of ProbLog

4/5: Testing our implementation's performance

Network "SMALL"

iterative sampling

path	983 – 620			983 – 582			620 – 582		
δ	S	T_p	P	S	T_p	P	S	T_p	P
0.1	1000	19	0.10	1000	21	0.10	1000	63	1.00
0.05	1000	19	0.10	1000	23	0.11	1000	59	1.00
0.01	16000	898	0.11	16000	1418	0.11	1000	59	1.00

Table 3. Monte Carlo Inference on SMALL.

* great results

  even 1st iteration

4/5: Testing our implementation's performance

Network "MEDIUM"

k-best

path	983 – 620			983 – 582			620 – 582		
k	T_p	T_B	P	T_p	T_B	P	T_p	T_B	P
1	208	-	0.07	737	-	0.03	45	-	0.42
2	172	1591	0.11	725	1560	0.03	44	1599	0.47
4	200	1681	0.16	757	1738	0.05	60	1464	0.72
8	217	1691	0.25	744	1538	0.06	80	1778	0.92
16	284	1756	0.33	725	1508	0.10	100	1825	0.99
32	628	1855	0.38	753	1570	0.15	144	1578	1.00
64	717	1653	0.41	809	1684	0.23	200	1801	1.00
128	749	1715	0.42	933	1890	0.30	296	1734	1.00
256	849	1600	0.55	1044	1513	0.49	405	1904	1.00
512	2352	1696	0.64	2880	1598	0.53	576	2496	1.00
1024	6208	1849	0.70	5032	1728	0.56	2549	52250	1.00

Table 4. k-probability on MEDIUM.

* P and T are higher (~11k nodes)
* **Accurate computation is impractical.**

4/5: Testing our implementation's performance

Network "MEDIUM"

bounded approximation

diverged on "MEDIUM" (DNF's binary decision tree too complex for CUDD)

---

iterative sampling

works great (although times grow fast as δ decreases.)

memo	983 – 620			983 – 582			620 – 582		
δ	S	T_p	P	S	T_p	P	S	T_p	P
0.1	1000	1319	0.77	1000	2364	0.76	1000	1878	1.00
0.05	2000	2682	0.76	2000	4766	0.76	1000	1805	1.00
0.01	29000	39687	0.76	29000	70183	0.77	1000	1970	1.00

Table 5. Monte Carlo Inference using `memopath/3` on MEDIUM.

4/5: Testing our implementation's performance

Network "BIOMINE"

k-best

path	$983-620$			$983-582$			$620-582$		
k	T_p	T_B	P	T_p	T_B	P	T_p	T_B	P
1	5,445	-	0.09	1,248	-	0.11	10,189	-	0.59
2	5,472	1,611	0.12	1,313	1,563	0.17	2,288	1,570	0.63
4	5,989	1,735	0.13	13,729	1,986	0.28	600	1,545	0.65
8	7,016	1,656	0.16	19,885	1,878	0.38	929	1,792	0.66
16	10,012	1,980	0.50	30,338	1,816	0.53	1,557	1,644	0.92
32	14,857	1,872	0.57	35,134	1,657	0.56	2,484	1,922	0.95
64	19,770	1,642	0.80	36,995	1,737	0.65	4,425	1,925	0.95
128	23,165	1,892	0.88	163,242	1,835	0.76	8,472	2,117	0.98
256	35,395	2,149	0.95	292,054	1,463	0.85	16,390	4,935	1.00
512	170,438	3,148	0.98	489,254	15,410	0.88	29,525	7,693	1.00
1024	346,742	609,700	0.99	767,968	97,818	0.93	49,952	102,366	1.00

Table 6. k-probability on BIOMINE.

* Millions of vertices / edges -> P is almost always 1

* good performance, though.

4/5: Testing our implementation's performance

Network "BIOMINE"

bounded approximation

diverged on "BIOMINE"

---

iterative sampling

isn't useful on "BIOMINE" (even one iteration takes too long)

memo	$983-620$			$983-582$			$582-620$		
δ	S	T_p	P	S	T_p	P	S	T_p	P
0.1	1000	2,714,781	1.00	1000	4,887,260	0.97	1000	4,709,921	0.99
0.05	1000	2,807,927	1.00	1000	4,769,216	0.98	1000	4,823,262	0.99
0.01	1000	2,686,881	1.00	4000	19,187,318	0.98	2000	9,406,026	0.99

Table 7. Monte Carlo Inference using `memopath/3` on BIOMINE.

5/5: Summary

- ProbLog is a probabilistic logic language
  - Uncertain information is encoded by assigning a probability to each fact.
  - A probabilistic extension of Prolog.
  - Natural for mining / learning in biological networks
- This paper presents an implementation of ProbLog
  - On top of YAP-Prolog
- Implementation approximates probabilities efficiently
  - In large databases too
  - Much faster than original ProbLog
  - The first PLP to execute queries on databases of this magnitude (?)
  - Will introduce new possibilities & applications

Questions?

Thanks for listening!