Probabilistic Logic Programming with cplint

Week 1, lecture 2: approximate inference

Fabrizio Riguzzi

Approximate Inference

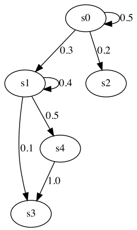
- For large models inference is intractable
- Approximate inference
  - Monte Carlo: draw samples of the truth value of the query (MCINTYRE)
  - Iterative deepening: gives a lower and an upper bound
  - Compute only the best *k* explanations: branch and bound, gives a lower bound

Monte Carlo

```
1:  function MONTECARLO($\mathcal{P}$, $q$, $n$)
2:      Input: Program $\mathcal{P}$, query $q$, number of samples $n$,
3:      Output: $P(q)$
4:      transform $\mathcal{P}$
5:      $Samples \leftarrow 0$
6:      $TrueSamples \leftarrow 0$
7:      for $i = 1 \rightarrow n$ do
8:          $Samples \leftarrow Samples + 1$
9:          if SAMPLE($q$) succeeds then
10:             $TrueSamples \leftarrow TrueSamples + 1$
11:         end if
12:     end for
13:     $\hat{p} \leftarrow \frac{TrueSamples}{Samples}$
14:     return $\hat{p}$
15: end function
```

Markov Chain Example:

Model checking of a Markov chain: we want to know what is the likelihood that on an execution of the chain from a start state s, a final state t will be reached?



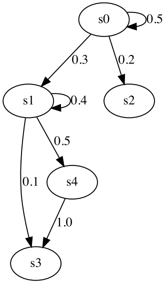
- The chains may be infinite so the query may have an infinite number of explanations
- PITA may not terminate.
- Two solutions. We may either fix a bound on the depth of the derivations of PITA by setting the parameters

```
:- set_pita(depth_bound,true).
:- set_pita(depth,<level of depth (integer)>).
```

Alternatively, MCINTYRE can be used.

Markov Chain Example:

```
% load the library 'mcintyre' to perform approximate inference
:- use_module(library(mcintyre)).
% load the renderer 'c3' for graphical results
:- use_rendering(c3).
% initialize the library 'mcintyre'
:- mc.
% to be written before the program
:- begin_lpad.
reach(S, I, T) :-
  trans(S, I, U),
  reach(U, next(I), T).
reach(S, _, S).
trans(s0,S,s0):0.5; trans(s0,S,s1):0.3; trans(s0,S,s2):0.2.
trans(s1,S,s1):0.4; trans(s1,S,s3):0.1; trans(s1,S,s4):0.5.
trans(s4,_,s3).
% to be written after the program
:- end_lpad.
```



MCINTYRE

To execute queries we must use the predicates `mc_prob/2`

```
mc_prob(:Query:atom,-Probability:float).
```

We ask for the probability that starting at state 's0' at instance 0, state 's3' is reachable

```
mc_prob(reach(s0,0,s3),P).
mc_prob(reach(s0,0,s3),P),bar(P,G).
```

MCINTYRE

You can also take a given number of sample with

```
mc_sample(:Query:atom,+Samples:int,
  -Probability:float).
```

The query

```
mc_sample(reach(s0,0,s3),1000,P).
```

samples `reach(s0,0,s3)` 1000 times and returns in `P` the estimated probability.

Sampling Arguments

We can also sample arguments of queries with the predicate `mc_sample_arg/4`

```
mc_sample_arg(:Query:atom,+Samples:int,
  ?Arg:var,-Values:list).
```

The predicate samples `Query` a number of `Samples` times.

- `Arg` should be a variable in `Query`.
- `Values` is a list of couples `L-N` where `L` is the list of values of `Arg` for which `Query` succeeds in world sampled at random and `N` is the number of samples.

Observations:

- If `L` is the empty list, it means that for that sample the query failed.
- If `L` is a list with a single element, it means that for that sample the query is determinate.

Sampling Arguments

```
mc_sample_arg(reach(s0,0,S),50,S,Values).
```

If we want to see the bar graph of this sampling we use the predicate `argbar/2`

```
argbar(+List:list,-Chart:dict).
```

For example

```
mc_sample_arg(reach(s0,0,S),50,S,List),
  argbar(List,Chart).
```

Sampling Arguments

Moreover, we can sample arguments of queries with the predicate `mc_sample_arg_first/4`

```
mc_sample_arg_first(:Query:atom,+Samples:int,
  ?Arg:var,-Values:list)
```

- `Values` is a list of couples `V-N` where `V` is the value of `Arg` returned as the first answer by `Query` in a world sampled at random and `N` is the number of samples returning that value. `V` is failure if the query fails.

Example

```
mc_sample_arg_first(reach(s0,0,S),50,S,Values).
```

Computing Expectations

Example: Coupon Collector Problem

`http://cplint.eu/e/coupon.swinb`

> *Suppose each box of cereal contains one of N different coupons and once a consumer has collected a coupon of each type, he can trade them for a prize. The aim of the problem is determining the average number of cereal boxes the consumer should buy to collect all coupon types, assuming that each coupon type occurs with the same probability in the cereal boxes.*
>
> *If there are 5 different coupons, what is the expected number of boxes I have to buy to get the prize?*

Computing Expectations

```
mc_expectation(:Query:atom,+N:int,?Arg:var,
  -Exp:float).
```

Example

```
mc_expectation(coupons(5,T),100,T,Exp).
```

Approximate Conditional Inference

Example: random arithmetic functions

`http://cplint.eu/e/arithm.pl`

> *This example generatively defines a random arithmetic function. The problem is to predict the value returned by the function given one or two couples of input-output, i.e., to compute a conditional probability.*
> *Sampling is necessary as queries have an infinite number of explanations*

Rejection Sampling

In rejection sampling, the evidence is first queried and, if it is successful, the query is asked in the same sample; otherwise, the sample is discarded

```
mc_rejection_sample(:Query:atom,:Evidence:atom,
+Samples:int,-Probability:float,+Options:list).
```

Example

```
mc_rejection_sample(eval(2,4),eval(1,3),1000,P,[]).
```

Metropolis-Hastings

- In Metropolis-Hastings Markov Chain Monte Carlo, a Markov chain is built by taking an initial sample and by generating successor samples
- After a sample, a number (*lag*) of sampled probabilistic choices are deleted and the others are retained for the next sample.
- The sample is accepted with a probability of $\min\{1, N0/N1\}$ where $N0$ is the number of choices sampled in the previous sample and $N1$ is the number of choices sampled in the current sample.
- Metropolis-Hastings is usually much faster than rejection sampling because less samples are discarded.

Metropolis-Hastings

To use Metropolis-Hastings, the following predicate is available

```
mc_mh_sample(:Query:atom,:Evidence:atom,+Samples:int,
  -Probability:float,+Options:list).
```

where `Options` is a list of options, the following are recognised:

- `mix(+Mix:int)` The first Mix samples are discarded (mixing time), default value 0
- `lag(+lag:int)` lag between each sample, Lag sampled choices are forgotten, default value 1
- `successes(-successes:int)` Number of successes
- `failures(-failures:int)` Number of failures

Metropolis-Hastings

```
mc_mh_sample(eval(2,4),eval(1,3),10000,P).
```

takes 10000 accepted samples and returns in P the estimated probability

Sampling Arguments

You can sample arguments of queries with rejection sampling and Metropolis-Hastings MCMC using

```
mc_rejection_sample_arg(:Query:atom,:Evidence:atom,
  +Samples:int,?Arg:Var,-Values:list,+Options:list).
mc_mh_sample_arg(:Query:atom,:Evidence:atom,
  +Samples:int,?Arg:Var,-Values:list,+Options:list).
```

Example

```
mc_mh_sample_arg(eval(2,Y),eval(1,3),1000,Y,L),
  argbar(L,C).
```

Conditional Expectations

```
mc_mh_expectation(:Query:atom,:Evidence:atom,+N:int,
  ?Arg:var,-Exp:float,+Options:list).
```

Example

```
mc_mh_expectation(eval(2,Y),eval(1,3),1000,Y,E,[]).
```