Probabilistic Programming

Maarten Sijmkens

August 2020

1 Probabilistic Inference Using Weighted Model Counting

1.1 1.1 SRL to CNF

First we ground the program to eliminate all variables. Next we break cycles and replace all probabilistic heads with a helper predicate with a given weight.

```
stress(a) :- p(1,a).
stress(b) :- p(1,b).
stress(c) :- p(1,c).

friends(a,b) :- p(2,a,b).
friends(a,c) :- p(2,a,c).
friends(b,c) :- p(2,b,c).
friends(c,b) :- p(2,c,b).

smokes(a) :- stress(a), p(3,a).
smokes(a) :- friends(a,b), smokes(b), p(4,a,b).
smokes(a) :- friends(a,c), smokes(c), p(4,a,c).

smokes_0(b) :- stress(b), p(3,b).
smokes(b) :- smokes_0(b).
smokes(b) :- friends(b,c), smokes_0(c), p(4,b,c).

smokes_0(c) :- stress(c), p(3,c).
smokes(c) :- smokes_0(c).
smokes(c) :- friends(c,b), smokes_0(b), p(4,c,b).

query(smokes(a)).

P(p(1,X)) = 0.2
P(p(2,X,Y)) = 0.1
P(p(3,X)) = 0.3
P(p(4,X,Y)) = 0.4
```

Finally we have to convert this logic formula to CNF. All rules with the same head can be converted to an equivalence using Clark's completion.
`a :- b` and `a :- c` results in $a \Leftrightarrow b \vee c$ under closed world assumption. The conversion of an equivalence to CNF is trivial. The cnf file is given in `program.cnf`. The associated weights are given in a comment line in the cnf file.

1.2 SRL to PGM

The figure gives an equivalent Bayesian Network for the problog program. This network is represented in `bayesian_network.py`. The CPT table of `smokes(a)` and `c6` are given below in table 1 and 2. The other

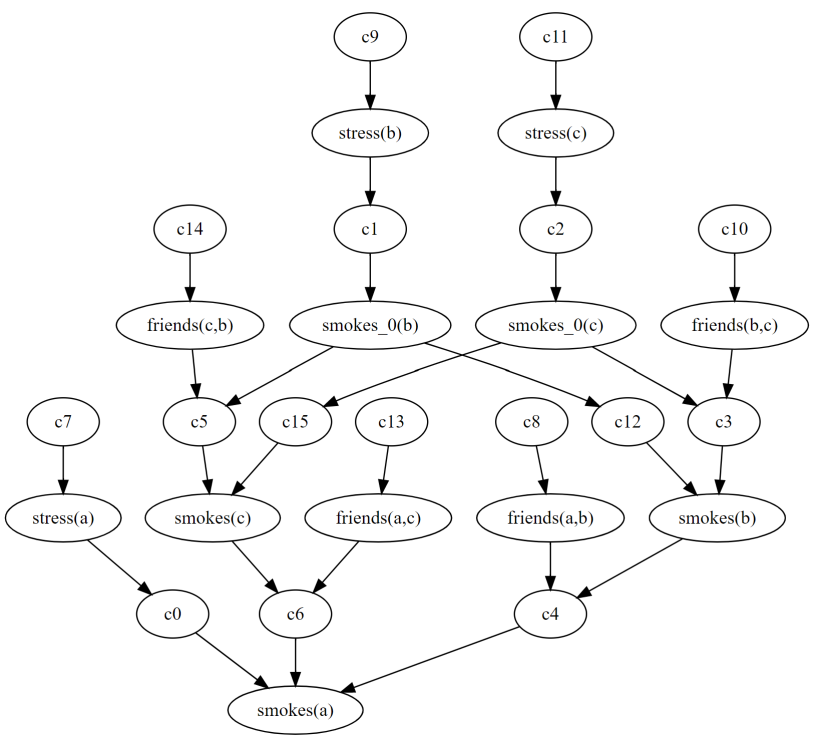

Figure 1: Bayesian network

tables are very similar and their probabilities can be inspected with the `print_CPT` function in the python file.

1.3 PGM to CNF

`to_cnf_enc1.py` converts the network to CNF with the enc1 method. It is found in `program_enc1.cnf`.

`to_cnf_enc2.py` converts the network with the enc2 method. It is found in `program_enc2.cnf`.

Both encodings take a similar approach. First, for every network variable, two literals are defined to represent the two values the variable can be. (true or false). Then the indicator clauses are defined so that no variable can take two values at the same time. Then for every conditional probability in the CPT of the networkvariable, the parameter clauses are defined. These parameter clauses are more compact in enc2.

Note that these python files can only work with network variables that can have two values (true or false), while the paper describes a method for a more general case where it can have multiple values.

1.4 Weighted Model Counting

Performing weighted model counting on this program we find for all encodings that `P(smokes(a)) = 0.065`.

SDD on `program.cnf`

```
compilation time        : 0.049 sec
sdd size                : 244
sdd node count          : 120
sdd model count         : 5800    0.000 sec
sdd weighted model count: 0.064669450752    0.000 sec
```

SDD on `program_enc1.cnf`

```
compilation time        : 1.534 sec
sdd size                : 2332
sdd node count          : 1078
sdd model count         : 134217728    0.000 sec
sdd weighted model count: 0.064669450752    0.000 sec
```

SDD on `program_enc2.cnf`

```
compilation time         : 0.879 sec
sdd size                : 2562
sdd node count          : 857
sdd model count         : 0    0.000 sec
sdd weighted model count: 0.06466945075200002    0.000 sec
```

To find `P(smokes(a)|friends(a,b),friends(a,c))` we use

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

In `to_cnf_cond.py` we generate 2 CNF files: `program_cond1.cnf` for `P(smokes(a),friends(a,b),friends(a,c))` and `program_cond2.cnf` for `P(friends(a,b),friends(a,c))`. We find `P(smokes(a)|friends(a,b),friends(a,c)) = 0.00106 / 0.01 = 0.106`. Alternatively we could set the weights of `friends(a,b)` and `friends(a,c)` on (1,0) and query `smokes(a)` using only one CNF.

SDD on `program_cond1.cnf`

```
compilation time        : 0.061 sec
sdd size                : 185
sdd node count          : 89
sdd model count         : 1816    0.000 sec
sdd weighted model count: 0.0010559646720000003    0.000 sec
```

SDD on `program_cond2.cnf`

```
compilation time        : 0.038 sec
sdd size                : 298
sdd node count          : 137
sdd model count         : 4096    0.000 sec
sdd weighted model count: 0.010000000000000004    0.000 sec
```

SDD uses an efficient representation of the boolean formula so that it can efficiently find models. Once a model is found, its probability can simply be calculated by multiplying the weights. The total probability is the sum of the probabilities of all models.

2 Lifted Inference

The goal is to find $P(Q_a)$ where Q_a is the query in the given program. We expand the query until it only contains predicates of which we know the probability. Then we calculate the probability using the lifted inference rules.

$Q_a = smokes(a)$
$Q_a = (stress(a) \wedge p(3,a)) \vee (friends(a,b) \wedge smokes(b) \wedge p(4,a,b)) \vee (friends(a,c) \wedge smokes(c) \wedge p(4,a,c))$
$Q_a = (p(1,a) \wedge p(3,a)) \vee (p(2,a,b) \wedge smokes(b) \wedge p(4,a,b)) \vee (p(2,a,c) \wedge smokes(c) \wedge p(4,a,c))$
$P(Q_a) = 1 - ((1 - P(p(1,a))P(p(3,a)))(1 - (1 - P(p(2,a,b))P(Q_b)P(p(4,a,b)))(1 - P(p(2,a,c))P(Q_c)P(p(4,a,c)))))$
$P(Q_a) = 1 - (1 - 0.2 * 0.3)(1 - (1 - (1 - 0.1 * 0.062256 * 0.4)(1 - 0.1 * 0.062256 * 0.4))) = 0.064675822$

$Q_b = smokes(b)$
$Q_b = (smokes\_0(b)) \vee (friends(b,c) \wedge smokes\_0(c) \wedge p(4,b,c))$
$Q_b = (smokes\_0(b)) \vee (p(2,b,c) \wedge smokes\_0(c) \wedge p(4,b,c))$
$P(Q_b) = 1 - ((1 - P(Q_{b0}))(1 - P(p(2,b,c))P(Q_{c0})P(p(4,b,c))))$
$P(Q_b) = 1 - ((1 - 0.06)(1 - 0.1 * 0.06 * 0.4)) = 0.062256$

$Q_c = smokes(c)$
$Q_c = (smokes\_0(c)) \vee (friends(c,b) \wedge smokes\_0(b) \wedge p(4,c,b))$
$Q_c = (smokes\_0(c)) \vee (p(2,c,b) \wedge smokes\_0(b) \wedge p(4,c,b))$
$P(Q_c) = 1 - ((1 - P(Q_{c0}))(1 - P(p(2,c,b))P(Q_{b0})P(p(4,c,b))))$
$P(Q_c) = 1 - ((1 - 0.06)(1 - 0.1 * 0.06 * 0.4)) = 0.062256$

$Q_{b0} = smokes\_0(b)$
$Q_{b0} = stress(b) \wedge p(3,b)$
$Q_{b0} = p(1,b) \wedge p(3,b)$
$P(Q_{b0}) = P(p(1,b))P(p(3,b))$
$P(Q_{b0}) = 0.2 * 0.3 = 0.06$

$Q_{c0} = smokes\_0(c)$
$Q_{c0} = stress(c) \wedge p(3,c)$
$Q_{c0} = p(1,c) \wedge p(3,c)$
$P(Q_{c0}) = P(p(1,c))P(p(3,c))$
$P(Q_{c0}) = 0.2 * 0.3 = 0.06$

3 Parameter learning

In every episode m we use probabilities $\hat{p}^{(m)}$ for the friends predicates. We initiate $\hat{p}^{(0)}$ arbitrarily. Now using these probabilities, for every friends predicate f_n and every interpretation I_m we calculate $P(f_n|I_m)$ using Bayes rule:

$$P(f_n|I_m) = \frac{P(I_m|f_n)P(f_n)}{P(I_m)}$$

For $P(f_n)$ we use the current estimate $\hat{p}^{(m)}$. $P(I_m)$ and $P(I_m|f_n)$ can be calculated using two weighted model counts. Since we have M interpretations and 6 friends predicates, we will need to calculate 2*6*M wmc's per episode. We can do this very efficiently in SDD: we only need to change the weights of the SDD model of the program. For the interpretations we need to set the weight of a predicate to (1,0) if there is positive evidence and (0,1) if it is negative. After every episode we can update the estimate $\hat{p}^{(m)}$ as stated in the assignment. If the estimates change with less than a value ϵ, we stop iterating. The algorithm is implemented in `learning.py`.

With $M = 2000$ and $\epsilon = 0.01$, we find:

```
friends(a,b) : 0.458239186163989
friends(a,c) : 0.5624119629494368
friends(b,a) : 0.4582232971074153
friends(b,c) : 0.7924391158694617
friends(c,a) : 0.5282968530560769
friends(c,b) : 0.8272499045510201
```

With $M = 100$ and $\epsilon = 0.01$, we find:

```
friends(a,b) : 0.585686732904326
friends(a,c) : 0.6296019150807017
friends(b,a) : 0.40456952085125986
friends(b,c) : 0.9452302548816242
friends(c,a) : 0.3216201096613196
friends(c,b) : 0.9488385167834779
```

The difference in both results can probably be explained by that the first 100 samples do not represent the distribution as well as the complete 2000.

c0	**c6**	**c4**	**Pr**
false	false	false	0.0
false	false	true	1.0
false	true	false	1.0
false	true	true	1.0
true	false	false	1.0
true	false	true	1.0
true	true	false	1.0
true	true	true	1.0

Table 1: CPT of smokes(a)

smokes(c)	**friends(a,c)**	**Pr**
false	false	0.0
false	true	0.0
true	false	0.0
true	true	0.4

Table 2: CPT of c6