Informatics 2D – Reasoning and Agents

Semester 2, 2019–2020

Alex Lascarides
alex@inf.ed.ac.uk

School of informatics

Lecture 25 – Approximate Inference in Bayesian Networks
17th March 2020

Where are we?

Last time ...

- Inference in Bayesian Networks
- Exact methods: enumeration, variable elimination algorithm
- Computationally intractable in the worst case

Today ...

- **Approximate Inference in Bayesian Networks**

Approximate inference in BNs

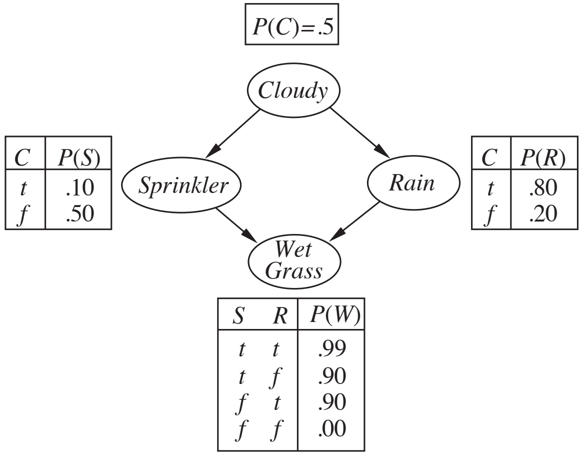
- Exact inference computationally very hard
- Approximate methods important, here randomised sampling algorithms
- **Monte Carlo** algorithms
- We will talk about two types of MC algorithms:
  1. Direct sampling methods
  2. Markov chain sampling

Direct sampling methods

- Basic idea: generate samples from a known probability distribution
- Consider an unbiased coin as a random variable – sampling from the distribution is like flipping the coin
- It is possible to sample any distribution on a single variable given a set of random numbers from [0,1]
- Simplest method: generate events from network without evidence
  - Sample each variable in 'topological order'
  - Probability distribution for sampled value is conditioned on values assigned to parents

Example

- Consider the following BN and ordering [*Cloudy*, *Sprinkler*, *Rain*, *WetGrass*]:



Example

- Direct sampling process:
  - Sample from $\mathbf{P}(Cloudy) = \langle 0.5, 0.5 \rangle$, suppose this returns *true*
  - Sample from $\mathbf{P}(Sprinkler|Cloudy = true) = \langle 0.1, 0.9 \rangle$, suppose this returns *false*
  - Sample from $\mathbf{P}(Rain|Cloudy = true) = \langle 0.8, 0.2 \rangle$, suppose this returns *true*
  - Sample from $\mathbf{P}(WetGrass|Sprinkler = false, Rain = true) = \langle 0.9, 0.1 \rangle$, suppose this returns *true*
- Event returned=[*true*, *false*, *true*, *true*]

Direct sampling methods

- Generates samples with probability $S(x_1, \ldots, x_n)$

$$S(x_1, \ldots, x_n) = P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$$

  i.e. in accordance with the distribution
- Answers are computed by counting the number $N(x_1, \ldots, x_n)$ of the times event $x_1, \ldots, x_n$ was generated and dividing by total number N of all samples
- In the limit, we should get

$$\lim_{n \to \infty} \frac{N(x_1, \ldots, x_n)}{N} = S(x_1, \ldots, x_n) = P(x_1, \ldots, x_n)$$

- If the estimated probability $\hat{P}$ becomes exact in the limit we call the estimate **consistent** and we write "$\approx$" in this sense, e.g. $P(x_1, \ldots, x_n) \approx N(x_1, \ldots, x_n)/N$

Rejection sampling

- Purpose: to produce samples for hard-to-sample distribution from easy-to-sample distribution
- To determine $P(X|\mathbf{e})$ generate samples from the prior distribution specified by the BN first
- Then reject those that do not match the evidence
- The estimate $\hat{P}(X = x|\mathbf{e})$ is obtained by counting how often $X = x$ occurs in the remaining samples
- Rejection sampling is consistent because, by definition:

$$\hat{P}(X|\mathbf{e}) = \frac{\mathbf{N}(X, \mathbf{e})}{N(\mathbf{e})} \approx \frac{\mathbf{P}(X, \mathbf{e})}{P(\mathbf{e})} = \mathbf{P}(X|\mathbf{e})$$

Back to our example

- Assume we want to estimate $\mathbf{P}(Rain|Sprinkler = true)$, using 100 samples
  - 73 have $Sprinkler = false$ (rejected), 27 have $Sprinkler = true$
  - Of these 27, 8 have $Rain = true$ and 19 have $Rain = false$
- $\mathbf{P}(Rain|Sprinkler = true) \approx \alpha\langle 8, 19\rangle = \langle 0.296, 0.704\rangle$
- True answer would be $\langle 0.3, 0.7\rangle$
- But the procedure rejects too many samples that are not consistent with $\mathbf{e}$ (exponential in number of variables)
- Not really usable (similar to naively estimating conditional probabilities from observation)

Likelihood weighting

- Avoids inefficiency of rejection sampling by generating only samples consistent with evidence
- Fixes the values for evidence variables $\mathbf{E}$ and samples only the remaining variables X and $\mathbf{Y}$
- Since not all events are equally probable, each event has to be weighted by its **likelihood** that it accords to the evidence
- Likelihood is measured by product of conditional probabilities for each evidence variable, given its parents

Likelihood weighting

- Consider query $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$ in our example; initially set weight $w = 1$, then event is generated:
  - Sample from $\mathbf{P}(Cloudy) = \langle 0.5, 0.5\rangle$, suppose this returns *true*
  - *Sprinkler* is evidence variable with value *true*, we set
    $$w \leftarrow w \times P(Sprinkler = true|Cloudy = true) = 0.1$$
  - Sample from $\mathbf{P}(Rain|Cloudy = true) = \langle 0.8, 0.2\rangle$, suppose this returns *true*
  - *WetGrass* is evidence variable with value *true*, we set
    $$w \leftarrow w \times P(WetGrass = true|Sprinkler = true, Rain = true) = 0.099$$
- Sample returned=$[true, true, true, true]$ with weight 0.099 tallied under $Rain = true$

Likelihood weighting – why it works

- $S(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i|parents(Z_i))$
- S's sample values for each Z_i is influenced by the evidence among Z_i's ancestors
- But S pays no attention when sampling Z_i's value to evidence from Z_i's non-ancestors; so it's not sampling from the true posterior probability distribution!
- But the likelihood weight w makes up for the difference between the actual and desired sampling distributions:

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i|parents(E_i))$$

Likelihood weighting – why it works

- Since two products cover all the variables in the network, we can write

$$P(\mathbf{z},\mathbf{e}) = \underbrace{\prod_{i=1}^{l} P(z_i|parents(Z_i))}_{S(\mathbf{z},\mathbf{e})} \underbrace{\prod_{i=1}^{m} P(e_i|parents(E_i))}_{w(\mathbf{z},\mathbf{e})}$$

- With this, it is easy to derive that likelihood weighting is consistent (tutorial exercise)
- Problem: most samples will have very small weights as the number of evidence variables increases
- These will be dominated by tiny fraction of samples that accord more than infinitesimal likelihood to the evidence

The Markov chain Monte Carlo (MCMC) algorithm

- MCMC algorithm: create an event from a previous event, rather than generate all events from scratch
- Helpful to think of the BN as having a **current state** specifying a value for each variable
- Consecutive state is generated by sampling a value for one of the non-evidence variables X_i conditioned on the current values of variables in the Markov blanket of X_i
- Recall that Markov blanket consists of parents, children, and children's parents
- Algorithm randomly wanders around state space flipping one variable at a time and keeping evidence variables fixed

The MCMC algorithm

- Consider query $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$ once more
- *Sprinkler* and *WetGrass* (evidence variables) are fixed to their observed values, hidden variables *Cloudy* and *Rain* are initialised randomly (e.g. *true* and *false*)
- Initial state is $[true, true, false, true]$
- Execute repeatedly:
  - Sample *Cloudy* given values of Markov blanket, i.e. sample from $\mathbf{P}(Cloudy|Sprinkler = true, Rain = false)$
  - Suppose result is *false*, new state is $[false, true, false, true]$
  - Sample *Rain* given values of Markov blanket, i.e. sample from $\mathbf{P}(Rain|Sprinkler = true, Cloudy = false, WetGrass = true)$
  - Suppose we obtain $Rain = true$, new state $[false, true, true, true]$

The MCMC algorithm – why it works

- Each state is a sample, contributes to estimate of query variable *Rain* (count samples to compute estimate as before)
- Basic idea of proof that MCMC is consistent:

  The sampling process settles into a "dynamic equilibrium" in which the long-term fraction of time spent in each state is exactly proportional to its posterior probability

- MCMC is a very powerful method used for all kinds of things involving probabilities

Summary

- Approximate inference in BN's
- Direct sampling methods
- Likelihood working and why it works
- MCMC algorithm and why it works
- Next time: **Time and Uncertainty I**