Inference in Bayesian Networks

- You have a Bayesian network.
  - Let \( Z = \{X_1, X_2, \ldots, X_n\} \) be a set of \( n \) discrete variables
- What do you do with it?
  - Queries
  - As we already know, the joint is modeled by

\[
P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \text{parents}(X_i))
\]
Conditioning

When we want to explain a complex event in terms of simpler events, we “condition”.

- Let $E \subseteq Z$, a set of instantiated variables.
- Let $X$ be the remaining variables in $Z$. Then,

Computing the probability of an event $E$

$$P(E) = \sum_{X} P(X, E)$$

$$P(E) = \sum_{X} \prod_{i=1}^{n} P(X_i \mid \text{parents}(X_i))$$

What is wrong?

- Solving the previous equation takes time which is exponential in $X$
  - We see this before we learn to “push in” summations.
- Just to store a Bayesian network takes room, depending on the connectivity of the network
  - More parents means more table entries in the CPTs.
- Bottom line: We have problems with time and space complexity.
Example

- You have two emotional states (H).
- You have a pet rabbit (R).
  - Happy: your pet rabbit is alive.
  - Sad: your pet rabbit is dead.

Example

- Your new neighbor is a crocodile farmer. If he farms (F), there is a risk of crocodile attack (C).
Example

- Your new neighbor is a crocodile farmer. If he farms (F), there is a risk of crocodile attack (C).

- The crocodile can eat your rabbit. You think you are scared of crocodile attacks.

\[
\begin{array}{cccc}
F & C & R & H \\
0.9 & 0.1 & 0.2 & 0.8 \\
0.1 & 0.9 & 1 & 0
\end{array}
\]

Example

- More parents = more space.
- If we want to compute \( P(H) \), the computer does this:

\[
P(H) = \sum_{F} \sum_{C} \sum_{R} P(F)P(C|F)P(R|C)P(H|R,C)
\]
Network Conditioning

- We can make things simpler if we condition the network on $C=c$ (being true).
- Cutset conditioning works to disconnect multiply-connected networks
  - Resulting singly-connected graph can be solved efficiently using poly-tree algorithms

Assume $C = c$
Network Conditioning

Assume C = c

We can save on space immediately – only half of the CPT for H is needed.

The network is now singly connected.
(Linear time and space complexity)
Network Conditioning

- We can make things simpler if we condition the network on \( C = c \) (being true).
- The result is a new, simpler network which allows any computation involving \( C = c \). Just as easily, another network can be created for \( C = \bar{c} \) and then we compute \( P(H) \) as the sum over conditions:

\[
P(H) = \sum_C \left[ \sum_F \sum_R P(F) P(C \mid F) P(R \mid C) P(H \mid R) \right]
\]

Network Decomposition

- Instead of worrying about single connectivity, it is easier to completely disconnect a graph into two subgraphs.
- Similar to tree-decomposition – which decomposition to pick?
  - We can use the BBN structure to decide
  - Any decomposition works, but some are more efficient than others.
D-trees

- D-Tree: full binary tree where leaves are network CPTs
- We should decompose the original network by instantiating variables shared by left and right branches

Decomposition

- Smaller, less-connected networks are along the nodes of the d-tree
Decomposition

The structure of the d-tree also shows how the computation can be factored.

Conditioning imposes independence between the variables in the factored portions of the graph.

Factoring

All inference tasks are sums of products of conditional probabilities.
Factoring

\[ P(H) = \sum_F \sum_C \sum_R P(F)P(C \mid F)P(R \mid C)P(H \mid R, C) \]

\[ = \sum_{FCR \in FCR} \prod_{i \in C} P(X_i \mid \text{parents}(X_i)) \]

\[ = \sum_C \left[ \sum_{FR \in FR} \prod_{i \in R} P(X_i \mid \text{parents}(X_i)) \right] \]

- All inference tasks are sums of products of conditional probabilities

Factoring

\[ P(H) = \sum_F \sum_C \sum_R P(F)P(C \mid F)P(R \mid C)P(H \mid R, C) \]

\[ = \sum_{FCR \in FCR} \prod_{i \in C} P(X_i \mid \text{parents}(X_i)) \]

\[ = \sum_C \left[ \sum_{FR \in FR} \prod_{i \in R} P(X_i \mid \text{parents}(X_i)) \right] \]

- At each step, you choose a new “cutset” and work with the subsequent networks
Recursive Conditioning Algorithm

Algorithm \textsc{rc1}
\[
\text{rc1}(T)
\begin{align*}
01. & \text{if } T \text{ is a leaf node,} \\
02. & \text{then return } \text{lookup}(T) \\
03. & \text{else } p \leftarrow 0 \\
04. & \text{for each instantiation } c \text{ of uninstantiated variables in } \text{cutset}(T) \text{ do} \\
05. & \hspace{1em} \text{record instantiation } c \\
06. & \hspace{1em} p \leftarrow p + \text{rc1}(T'c) \text{rc1}(T'c)' \\
07. & \hspace{1em} \text{un-record instantiation } c \\
08. & \text{return } p
\end{align*}
\]

\text{lookup}(T)
\[
\begin{align*}
01. & \phi \leftarrow \text{CPT of variable } X \text{ associated with leaf } T \\
02. & \text{if } X \text{ is instantiated,} \\
03. & \text{then } x \leftarrow \text{recorded instantiation of } X \\
04. & p \leftarrow \text{recorded instantiation of } X \text{'s parents} \\
05. & \text{return } \phi(x \mid p) \quad \text{// } \phi(x \mid p) = p(x \mid p) \\
06. & \text{else return } 1
\end{align*}
\]

Cutsets

- Conditioning on cutsets allow us to decompose the graph.
- \( \text{cutset}(T) = \text{var } s(T_L) \cap \text{var } s(T_R) - \text{acutset}(T) \)
- \( \text{acutset}(T) = \text{The union of all cutsets associated with } T \text{'s ancestor nodes} \)
Cutsets

A-Cutsets
Some intuition

- A cutset tells us what we are conditioning on.
- An A-cutset represents all of the variables being instantiated at that point on the d-tree.
  - We produce a solution for a subtree for every possible instantiation of the variables in the subtree’s A-cutset.
  - There can be redundant computation.

Contexts

- Several variables in the acutset may never be used in the subtree.
**Contexts**

- Several variables in the acutset may never be used in the subtree.
- We can instead remember the “context” under which any pair of computations yields the same result.

\[
\text{context}(T) = \text{vars}(T) \cap \text{acutset}(T)
\]

**Improved Recursive Conditioning Algorithm**

```
Algorithm rc2(T)
01. if T is a leaf node, 
02. then return \text{LOOKUP}(T)
03. else y ← \text{recorded instantiation of context}(T)
04. if cache[y] \neq \text{null}, return \text{cache}[y]
05. else p ← 0
06. for each instantiation c of uninstantiated variables in \text{cutset}(T) do
07. record instantiation c 
08. p ← p + \text{rc2}(T^c)\text{rc2}(T^c)
09. un-record instantiation c
10. cache[y] ← p
11. return p
```
Relation to Junction-Trees

- Sepsets are equivalent to contexts
- Messages passed between links correspond to contextual information being passed upward in the d-tree
- Passed messages sum out information about a residual (eliminated) set of variables – this is equivalent to the cutset.
- A d-tree can be built from a tree decomposition

Summary

- RC operates in $O(n \exp(w))$ time if you cache every context. This is better than being exponential in $n$.
- Caching can be selective, allowing the algorithm to run with limited memory
- Eliminates redundant computation
- Intuitively solves a complex event in terms of smaller events.