Belief network inference

Main approaches to determine posterior distributions in graphical models, depending on guarantees:

- Exact inference: exploit the structure of the network to eliminate (sum out) the non-observed, non-query variables one at a time (recursive conditioning, variable elimination).
- Guaranteed bounds of the conditional probabilities from above and below, where bound becomes narrower with more computation.
- Probabilistic bounds, e.g., within 0.1 of the correct answer 95% of the time. Stochastic simulation: random cases are generated according to the probability distributions.
- Best effort to produce an approximation that may be good enough. Variational methods: find the closest tractable distribution to the target (posterior) distribution.

Queries and Evidence

- To compute the posterior probability of query Q given evidence $E = e$:

$$
\begin{aligned}
& P(Q \mid E = e) \\
= & \frac{P(Q, E = e)}{P(E = e)} \\
= & \frac{P(Q, E = e)}{\sum_Q P(Q, E = e).}
\end{aligned}
$$

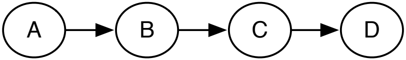
  summing over the values of variable Q
- So the computation reduces to the probability of $P(Q, E = e)$
- then normalize at the end.

Probability of a conjunction

- The variables of the belief network are $X_1, \ldots, X_n$.
- The evidence is $Y_1 = v_1, \ldots, Y_j = v_j$
- To compute $P(Q, Y_1 = v_1, \ldots, Y_j = v_j)$:
  add the other variables,
  $Z_1, \ldots, Z_k = \{X_1, \ldots, X_n\} - \{Q\} - \{Y_1, \ldots, Y_j\}$.
  and sum them out.
- Order the Z_i into an elimination ordering.

$$
\begin{aligned}
& P(Q, Y_1 = v_1, \ldots, Y_j = v_j) \\
&= \sum_{Z_k} \cdots \sum_{Z_1} P(X_1, \ldots, X_n)_{Y_1 = v_1, \ldots, Y_j = v_j}. \\
&= \sum_{Z_k} \cdots \sum_{Z_1} \prod_{i=1}^{n} P(X_i \mid parents(X_i))_{Y_1 = v_1, \ldots, Y_j = v_j}.
\end{aligned}
$$

Example



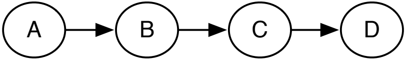
Query $P(D)$.

$$P(D) = \sum_A \sum_B \sum_C P(A, B, C, D)$$
$$= \sum_A \sum_B \sum_C P(A) P(B \mid A) P(C \mid B) P(D \mid C)$$

Can be simplified to:

$$P(D) = \sum_A P(A) \sum_B P(B \mid A) \sum_C P(C \mid B) P(D \mid C)$$

Example: different elimination ordering



Query $P(D)$.

$$
\begin{aligned}
P(D) &= \sum_C \sum_B \sum_A P(A, B, C, D) \\
&= \sum_C \sum_B \sum_A P(A)P(B \mid A)P(C \mid B)P(D \mid C) \\
&= \sum_C P(D \mid C) \sum_B P(C \mid B) \sum_A P(A)P(B \mid A)
\end{aligned}
$$

Naive Search Algorithm

- Computes the value of summing variables from a product of factors

Input:

- *Con* – a context – an assignment of a value to some of the variables
- *Fs* – a set of factors (functions of variables)

Output: value summing out the unassigned variables:

$$\sum_{X_1 \dots X_k} \prod Fs$$

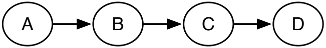
where $X_1 \dots X_k$ are variables not assigned in *Con*

- Evaluate a factor as soon as all its variables are assigned
- Recursively branch on a variable not assigned in *Con*
- Intially: *Con* is observations, and an assigment to the query variable, *Fs* is all the factors.

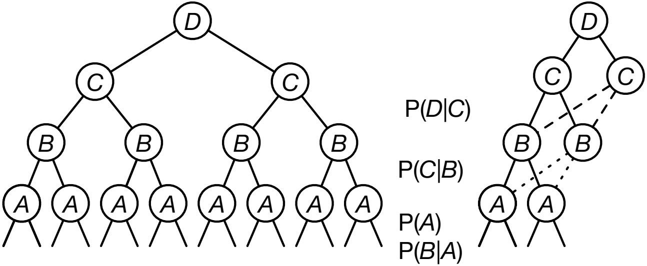
Probabilistic Inference using Depth-first search

```
procedure prob_dfs(Con : context, Fs : factors)
    if Fs = {} then
        return 1
    else if f ∈ Fs can be evaluated in Con then
        return eval(f, Con) ∗ prob_dfs(Con, Fs \ {f})
    else
        select variable X in not assigned in Con
        sum := 0
        for val in domain(X) do
            sum := sum + prob_dfs(Con ∪ {X=val}, Fs)
        return sum
```

Search Tree



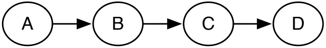
$$P(D) = \sum_C P(D \mid C) \sum_B P(C \mid B) \sum_A P(A) P(B \mid A)$$



(a) (b) (c)

Note: $\sum_B P(C \mid B) \sum_A P(A) P(B \mid A)$ does not depend on D
$\sum_A P(A) P(B \mid A)$ does not depend on C or D

Decomposition



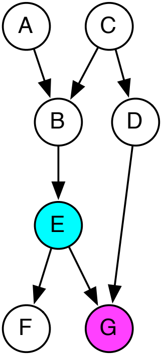
$$P(B \mid d)$$

First split on B (to compute the normalizing constant):

$$prob\_dfs(\{B{=}false, D{=}true\},$$
$$\{P(D \mid C), P(C \mid B), P(B \mid A), P(A)\})$$

This can be decomposed into two independent problems:

$$prob\_dfs(\{B{=}false, D{=}true\}, \{P(D \mid C), P(C \mid B)\})$$
$$* \; prob\_dfs(\{B{=}false, D{=}true\}, \{P(B \mid A), P(A)\})$$



$$P(E \mid g) = \frac{P(E \wedge g)}{\sum_E P(E \wedge g)}$$

$$P(E \wedge g)$$

$$= \sum_F \sum_B \sum_C \sum_A \sum_D P(A)P(B \mid AC) P(C)P(D \mid C)P(E \mid B)P(F \mid E)P(g \mid ED)$$

$$= \left(\sum_F P(F \mid E)\right) \sum_B P(E \mid B) \sum_C \left(P(C) \left(\sum_A P(A)P(B \mid AC)\right) \left(\sum_D P(D \mid C)P(g \mid ED)\right)\right)$$

Recursive Conditioning

Adds to naive search:

- Recognize when an assignment decomposes the problem into independent subproblems.
- Cache already computed values. The cache is checked before evaluating any query.

Recursive Conditioning

- Computes sum from outside in

Input:

- Context - assignment of values to variables
- Set of factors

Output: value of summing out other variables

Outline:

- Evaluate a factor as soon as all its variables are assigned
- Cache values already computed
- Recognize disconnected components
- Recursively branch on a variable

Primitive operations for the algorithm

- *cache* is a global variable that maps $\langle Con, Fs \rangle$ pairs into a real value.
  It is initially it has $\langle \{\}, \{\} \rangle$ mapping to 1.
- $vars(F)$ returns the variables in factor F.
  $vars(Fs)$ returns the variables that appear in any factor in Fs.
- $eval(F, Con)$ is the value of F given context Con. It is only called when Con assigns values to all of the variables in F.
- $Fs = Fs_1 \uplus Fs_2$ is the disjoint union, meaning $Fs_1 \neq \{\}$, $Fs_2 \neq \{\}$, $Fs_1 \cap Fs_2 = \{\}$, $Fs = Fs_1 \cup Fs_2$
  This step recognizes when the graph is disconnected.

Recursive Conditioning

procedure $rc(Con : \text{context}, \quad Fs : \text{set of factors})$:

```
if Fs = {} return 1
if ⟨Con, Fs⟩ is in cache with value v                          Recall
    return v
else if there is a variable in Con that is not in any factor in Fs
     return rc({X = v ∈ Con : X ∈ vars(Fs)}, Fs)     Forget
else if f ∈ Fs can be evaluated in Con
    return eval(f, Con) × rc(Con, Fs \ {f})                     Evaluate
else if Fs = Fs1 ⊎ Fs2 where vars(Fs1) ∩ vars(Fs2) are all assigned
    return rc(Con, Fs1) × rc(Con, Fs2)                          Disconnected
else select variable X ∈ vars(Fs) \ vars(Con)
    sum := 0
    for each v ∈ domain(X)
        sum := sum + rc(Con ∪ {X = v}, Fs)
    add ⟨Con, Fs⟩ with value sum to cache                       Remember
    return sum
```

Exploiting Structure in Recursive Conditioning

- How can we exploit determinism (zero probabilities)?
- How can we exploit context-specific independencies; the structure of decision trees or rules?
- How can we handle various representations of conditional probabilities / factors