Inference and Representation

David Sontag

New York University

Lecture 3, Sept. 26, 2016

Today's lecture

- Exact inference
  1. Worst-case complexity of probabilistic inference
  2. Elimination algorithm
  3. Running-time analysis of elimination algorithm (*treewidth*)
- Approximate inference

Probabilistic inference

- Today we consider exact inference in graphical models
- In particular, we focus on conditional probability queries,

$$p(\mathbf{Y}|\mathbf{E}=\mathbf{e}) = \frac{p(\mathbf{Y},\mathbf{e})}{p(\mathbf{e})}$$

  (e.g., the probability of a patient having a disease given some observed symptoms)

- Let $\mathbf{W} = \mathcal{X} - \mathbf{Y} - \mathbf{E}$ be the random variables that are neither the query nor the evidence. Each of these joint distributions can be computed by marginalizing over the other variables:

$$p(\mathbf{Y},\mathbf{e}) = \sum_{\mathbf{w}} p(\mathbf{Y},\mathbf{e},\mathbf{w}), \quad p(\mathbf{e}) = \sum_{\mathbf{y}} p(\mathbf{y},\mathbf{e})$$

- Naively marginalizing over all unobserved variables requires an exponential number of computations
- Does there exist a more efficient algorithm?

Computational complexity of probabilistic inference

- Here we show that, unless P=NP, there does *not* exist a more efficient algorithm
- We show this by reducing 3-SAT, which is NP-hard, to probabilistic inference in Bayesian networks
- 3-SAT asks about the *satisfiability* of a logical formula defined on n literals $Q_1, \ldots, Q_n$, e.g.

$$(\neg Q_3 \vee \neg Q_2 \vee Q_3) \wedge (Q_2 \vee \neg Q_4 \vee \neg Q_5) \cdots$$

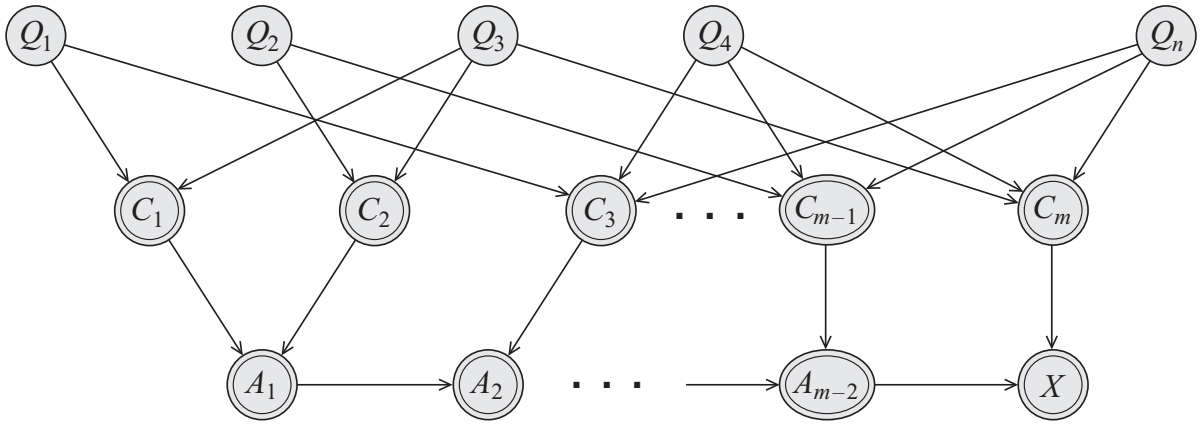
- Each of the disjunction terms is called a *clause*, e.g.

$$C_1(q_1, q_2, q_3) = \neg q_3 \vee \neg q_2 \vee q_3$$

  In 3-SAT, each clause is defined on at most 3 literals.
- Our reduction also proves that inference in Markov networks is NP-hard (why?)

Reducing satisfiability to MAP inference

- **Input:** 3-SAT formula with n literals $Q_1, \ldots Q_n$ and m clauses $C_1, \ldots, C_m$



- One variable $Q_i \in \{0, 1\}$ for each literal, $p(Q_i = 1) = 0.5$.
- One variable $C_i \in \{0, 1\}$ for each clause, whose parents are the literals used in the clause. $C_i = 1$ if the clause is satisfied, and 0 otherwise:

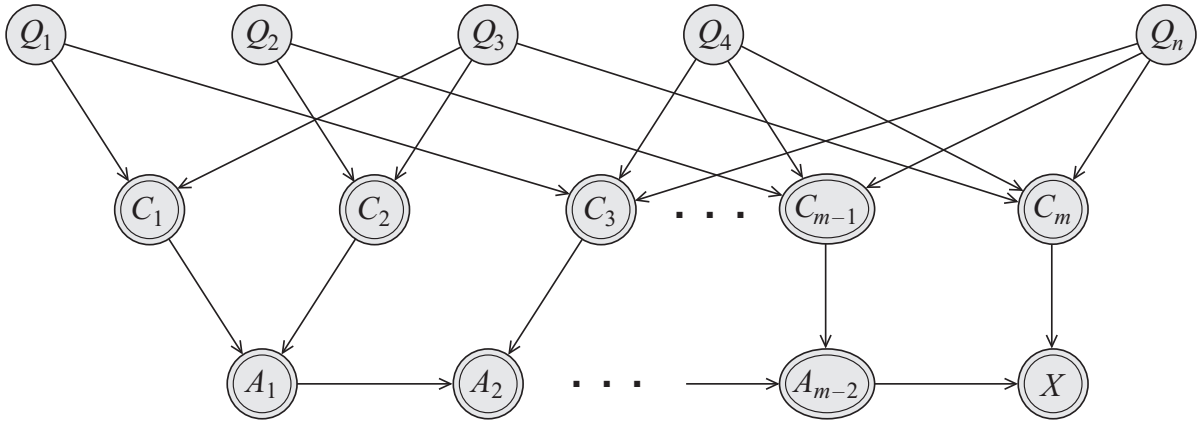
$$p(C_i = 1 \mid \mathbf{q}_{\mathrm{pa}(i)}) = 1[C_i(\mathbf{q}_{\mathrm{pa}(i)})]$$

- Variable X which is 1 if all clauses satisfied, and 0 otherwise:

$$\begin{aligned} p(A_i = 1 \mid \mathbf{pa}(A_i)) &= 1[\mathbf{pa}(A_i) = \mathbf{1}], \text{ for } i = 1, \ldots, m-2 \\ p(X = 1 \mid a_{m-2}, c_m) &= 1[a_{m-2} = 1, c_m = 1] \end{aligned}$$

Reducing satisfiability to MAP inference

- **Input:** 3-SAT formula with n literals $Q_1, \ldots Q_n$ and m clauses $C_1, \ldots, C_m$



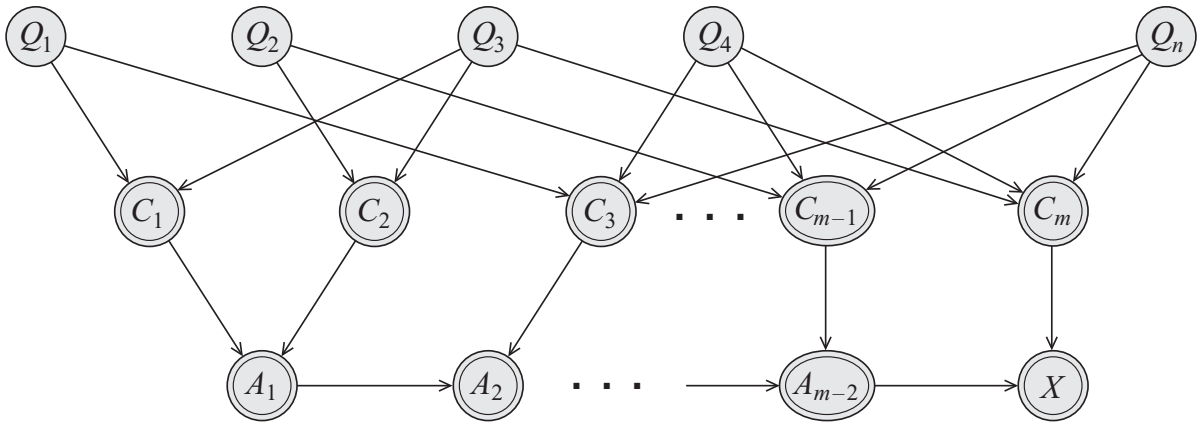
- $p(\mathbf{q}, \mathbf{c}, \mathbf{a}, X = 1) = 0$ for any assignment $\mathbf{q}$ which does not satisfy all clauses
- $p(\mathbf{Q} = \mathbf{q}, \mathbf{C} = \mathbf{1}, \mathbf{A} = \mathbf{1}, X = 1) = \frac{1}{2^n}$ for any satisfying assignment $\mathbf{q}$
- Thus, we can find a satisfying assignment (whenever one exists) by constructing this BN and finding the maximum a posteriori (MAP) assignment:

$$\operatorname*{argmax}_{\mathbf{q},\mathbf{c},\mathbf{a}} p(\mathbf{Q} = \mathbf{q}, \mathbf{C} = \mathbf{c}, \mathbf{A} = \mathbf{a} \mid X = 1)$$

- This proves that MAP inference in Bayesian networks and MRFs is NP-hard

Reducing satisfiability to marginal inference

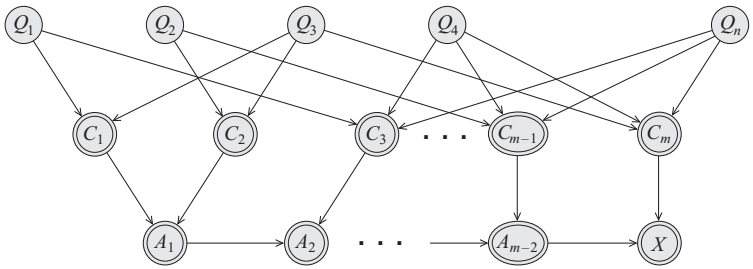
- **Input:** 3-SAT formula with n literals $Q_1, \ldots Q_n$ and m clauses $C_1, \ldots, C_m$



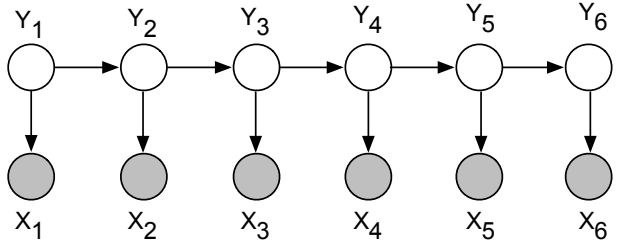
- $p(X = 1) = \sum_{\mathbf{q},\mathbf{c},\mathbf{a}} p(\mathbf{Q} = \mathbf{q}, \mathbf{C} = \mathbf{c}, \mathbf{A} = \mathbf{a}, X = 1)$ is equal to the number of satisfying assignments times $\frac{1}{2^n}$
- Thus, $p(X = 1) > 0$ if and only if the formula has a satisfying assignment
- This shows that *marginal inference* is also NP-hard

Probabilistic inference in practice

- NP-hardness simply says that there **exist** difficult inference problems
- Real-world inference problems are not necessarily as hard as these worst-case instances
- The reduction from SAT created a very complex Bayesian network:



- Some graphs are **easy** to do inference in! For example, inference in hidden Markov models



and other tree-structured graphs can be performed in **linear time**

Variable elimination (VE)

- Exact algorithm for probabilistic inference in **any** graphical model
- Running time will depend on the *graph structure*
- Uses **dynamic programming** to circumvent enumerating all assignments
- First we introduce the concept for computing marginal probabilities, $p(X_i)$, in Bayesian networks
- After this, we will generalize to MRFs and conditional queries

Basic idea

- Suppose we have a simple chain, $A \to B \to C \to D$, and we want to compute $p(D)$
- $p(D)$ is a **set** of values, $\{p(D = d), d \in \mathrm{Val}(D)\}$. Algorithm computes sets of values at a time – an entire distribution
- By the chain rule and conditional independence, the joint distribution factors as

$$p(A, B, C, D) = p(A)p(B \mid A)p(C \mid B)p(D \mid C)$$

- In order to compute $p(D)$, we have to marginalize over A, B, C:

$$p(D) = \sum_{a,b,c} p(A = a, B = b, C = c, D)$$

Let's be a bit more explicit...

$$
\begin{array}{rllll}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^1 \mid c^2)
\end{array}
$$

$$
\begin{array}{rllll}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}
$$

- There is structure to the summation, e.g., repeated $P(c^1|b^1)P(d^1|c^1)$
- Let's modify the computation to first compute

$$P(a^1)P(b^1|a^1) + P(a^2)P(b^1|a^2)$$

Let's be a bit more explicit...

- Let's modify the computation to first compute

$$P(a^1)P(b^1|a^1) + P(a^2)P(b^1|a^2)$$

and

$$P(a^1)P(b^2|a^1) + P(a^2)P(b^2|a^2)$$

- Then, we get

$$\begin{array}{lll}
& (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) \quad P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) \quad P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) \quad P(d^1 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) \quad P(d^1 \mid c^2)
\end{array}$$

$$\begin{array}{lll}
& (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) \quad P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) \quad P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) \quad P(d^2 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) \quad P(d^2 \mid c^2)
\end{array}$$

- We define $\tau_1 : \mathrm{Val}(B) \to \Re, \quad \tau_1(b^i) = P(a^1)P(b^i|a^1) + P(a^2)P(b^i|a^2)$

Let's be a bit more explicit...

- We now have

$$\begin{array}{lll} & \tau_1(b^1) & P(c^1 \mid b^1) \quad P(d^1 \mid c^1) \\ + & \tau_1(b^2) & P(c^1 \mid b^2) \quad P(d^1 \mid c^1) \\ + & \tau_1(b^1) & P(c^2 \mid b^1) \quad P(d^1 \mid c^2) \\ + & \tau_1(b^2) & P(c^2 \mid b^2) \quad P(d^1 \mid c^2) \end{array}$$

$$\begin{array}{lll} & \tau_1(b^1) & P(c^1 \mid b^1) \quad P(d^2 \mid c^1) \\ + & \tau_1(b^2) & P(c^1 \mid b^2) \quad P(d^2 \mid c^1) \\ + & \tau_1(b^1) & P(c^2 \mid b^1) \quad P(d^2 \mid c^2) \\ + & \tau_1(b^2) & P(c^2 \mid b^2) \quad P(d^2 \mid c^2) \end{array}$$

- We can once more reverse the order of the product and the sum and get

$$\begin{array}{lll} & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^1 \mid c^1) \\ + & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^1 \mid c^2) \end{array}$$

$$\begin{array}{lll} & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^2 \mid c^1) \\ + & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^2 \mid c^2) \end{array}$$

- There are still other repeated computations!

Let's be a bit more explicit...

- We define $\tau_2 : \mathrm{Val}(C) \to \Re$, with

$$\begin{aligned} \tau_2(c^1) &= \tau_1(b^1)P(c^1|b^1) + \tau_1(b^2)P(c^1|b^2) \\ \tau_2(c^2) &= \tau_1(b^1)P(c^2|b^1) + \tau_1(b^2)P(c^2|b^2) \end{aligned}$$

- Now we can compute the marginal $p(D)$ as

$$\begin{array}{lll} & \tau_2(c^1) & P(d^1 \mid c^1) \\ + & \tau_2(c^2) & P(d^1 \mid c^2) \end{array}$$

$$\begin{array}{lll} & \tau_2(c^1) & P(d^2 \mid c^1) \\ + & \tau_2(c^2) & P(d^2 \mid c^2) \end{array}$$

What did we just do?

- Our goal was to compute

$$p(D) = \sum_{a,b,c} p(a,b,c,D) = \sum_{a,b,c} p(a)p(b \mid a)p(c \mid b)p(D \mid c)$$
$$= \sum_{c}\sum_{b}\sum_{a} p(D \mid c)p(c \mid b)p(b \mid a)p(a)$$

- We can push the summations inside to obtain:

$$p(D) = \sum_{c} p(D \mid c) \sum_{b} p(c \mid b) \underbrace{\sum_{a} \underbrace{p(b \mid a)p(a)}_{\psi_1(a,b)}}_{\tau_1(b)}$$

- Let's call $\psi_1(A,B) = P(A)P(B|A)$. Then, $\tau_1(B) = \sum_a \psi_1(a,B)$
- Similarly, let $\psi_2(B,C) = \tau_1(B)P(C|B)$. Then, $\tau_2(C) = \sum_b \psi_1(b,C)$
- This procedure is dynamic programming: computation is inside out instead of outside in

Inference in a chain

- Generalizing the previous example, suppose we have a chain $X_1 \to X_2 \to \cdots \to X_n$ where each variable has k states
- In Problem Set 1 (question 2), you gave an algorithm to compute $p(X_i)$, for $k = 2$
- For $i = 1$ up to $n - 1$, compute (and cache)

$$p(X_{i+1}) = \sum_{x_i} p(X_{i+1} \mid x_i) p(x_i)$$

- Each update takes k^2 time (why?)
- The total running time is $\mathcal{O}(nk^2)$
- In comparison, naively marginalizing over all latent variables has complexity $\mathcal{O}(k^n)$
- We did inference over the joint without ever explicitly constructing it!

Summary so far

- Worst-case analysis says that marginal inference is NP-hard
- In practice, due to the structure of the Bayesian network, we can cache computations that are otherwise computed exponentially many times
- This depends on our having a good **variable elimination ordering**

Sum-product inference task

- We want to give an algorithm to compute $p(\mathbf{Y})$ for BNs and MRFs
- This can be reduced to the following **sum-product** inference task:

$$\text{Compute} \quad \tau(\mathbf{y}) = \sum_{\mathbf{z}} \prod_{\phi \in \Phi} \phi(\mathbf{z}_{\text{Scope}[\phi] \cap \mathbf{Z}}, \mathbf{y}_{\text{Scope}[\phi] \cap \mathbf{Y}}) \quad \forall \mathbf{y},$$

  where Φ is a set of factors or potentials

- For a BN, Φ is given by the conditional probability distributions for all variables,

$$\Phi = \{\phi_{X_i}\}_{i=1}^n = \{p(X_i \mid \mathbf{X}_{\text{Pa}(X_i)})\}_{i=1}^n,$$

  and where we sum over the set $\mathbf{Z} = \mathcal{X} - \mathbf{Y}$

- For Markov networks, the factors Φ correspond to the set of potentials which we earlier called C
  - Sum-product returns an unnormalized distribution, so we divide by $\sum_{\mathbf{y}} \tau(\mathbf{y})$

Factor marginalization

- Let $\phi(\mathbf{X}, Y)$ be a factor where $\mathbf{X}$ is a set of variables and $Y \notin \mathbf{X}$
- **Factor marginalization** of ϕ over Y (also called "summing out Y in ϕ") gives a new factor:

$$\tau(\mathbf{X}) = \sum_{Y} \phi(\mathbf{X}, Y)$$

For example,

a^1	b^1	c^1	0.25
a^1	b^1	c^2	0.35
a^1	b^2	c^1	0.08
a^1	b^2	c^2	0.16
a^2	b^1	c^1	0.05
a^2	b^1	c^2	0.07
a^2	b^2	c^1	0
a^2	b^2	c^2	0
a^3	b^1	c^1	0.15
a^3	b^1	c^2	0.21
a^3	b^2	c^1	0.09
a^3	b^2	c^2	0.18

a^1	c^1	0.33
a^1	c^2	0.51
a^2	c^1	0.05
a^2	c^2	0.07
a^3	c^1	0.24
a^3	c^2	0.39

Sum-product variable elimination

- Order the variables $\mathbf{Z}$ (called the **elimination ordering**)
- Iteratively marginalize out variable Z_i, one at a time
- For each i,
  1. Multiply all factors that have Z_i in their scope, generating a new product factor
  2. Marginalize this product factor over Z_i, generating a smaller factor
  3. Remove the old factors from the set of all factors, and add the new one

Algorithm 9.1 Sum-Product Variable Elimination algorithm

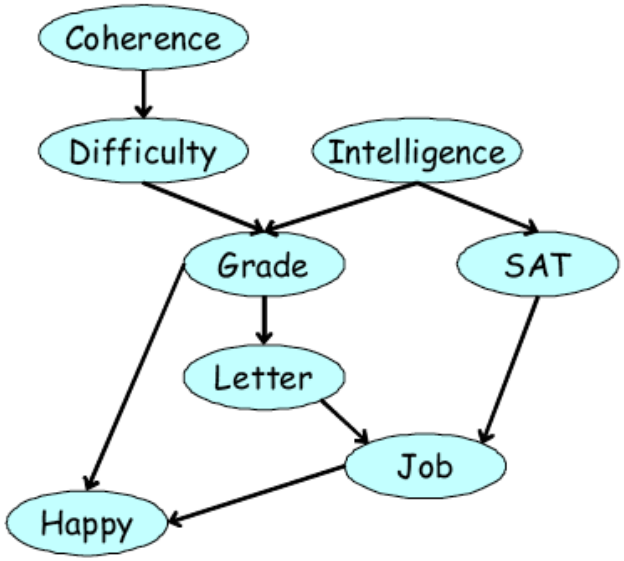
Procedure Sum-Product-Variable-Elimination (
  Φ, // Set of factors
  $\boldsymbol{Z}$, // Set of variables to be eliminated
  $\prec$ // Ordering on $\boldsymbol{Z}$
)

Let $Z_1, \ldots, Z_k$ be an ordering of $\boldsymbol{Z}$ such that
  $Z_i \prec Z_j$ iff $i < j$
for $i = 1, \ldots, k$
  $\Phi \leftarrow$ Sum-Product-Eliminate-Var(Φ, Z_i)
$\phi^* \leftarrow \prod_{\phi \in \Phi} \phi$
return ϕ^*

Procedure Sum-Product-Eliminate-Var (
  Φ, // Set of factors
  Z // Variable to be eliminated
)

$\Phi' \leftarrow \{\phi \in \Phi : Z \in Scope[\phi]\}$
$\Phi'' \leftarrow \Phi - \Phi'$
$\psi \leftarrow \prod_{\phi \in \Phi'} \phi$
$\tau \leftarrow \sum_Z \psi$
return $\Phi'' \cup \{\tau\}$

Example



- What is $p(\text{Job})$? Joint distribution factorizes as:

$$p(C, D, I, G, S, L, H, J) = p(C)p(D|C)p(I)p(G|D, I)p(L|G)P(S|I)P(J|S, L)p(H|J, G)$$

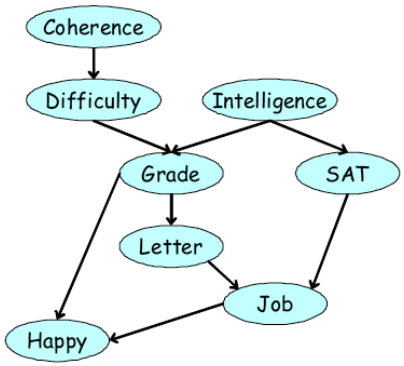
with factors

$$\Phi = \{\phi_C(C), \phi_D(C, D), \phi_I(I), \phi_G(G, D, I), \phi_L(L, G), \phi_S(S, I), \phi_J(J, S, L), \phi_H(H, J, G)\}$$

- Let's do variable elimination with ordering $\{C, D, I, H, G, S, L\}$ on the board!

Elimination ordering

- We can pick any order we want, but some orderings introduce factors with much larger scope



Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$

- Alternative ordering...

Step	Variable eliminated	Factors used	Variables involved	New factor
1	G	$\phi_G(G,I,D)$, $\phi_L(L,G)$, $\phi_H(H,G,J)$	G,I,D,L,J,H	$\tau_1(I,D,L,J,H)$
2	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_1(I,D,L,S,J,H)$	S,I,D,L,J,H	$\tau_2(D,L,S,J,H)$
3	S	$\phi_J(J,L,S)$, $\tau_2(D,L,S,J,H)$	D,L,S,J,H	$\tau_3(D,L,J,H)$
4	L	$\tau_3(D,L,J,H)$	D,L,J,H	$\tau_4(D,J,H)$
5	H	$\tau_4(D,J,H)$	D,J,H	$\tau_5(D,J)$
6	C	$\tau_5(D,J)$, $\phi_D(D,C)$	D,J,C	$\tau_6(D,J)$
7	D	$\tau_6(D,J)$	D,J	$\tau_7(J)$

How to introduce evidence?

- Recall that our original goal was to answer *conditional* probability queries,

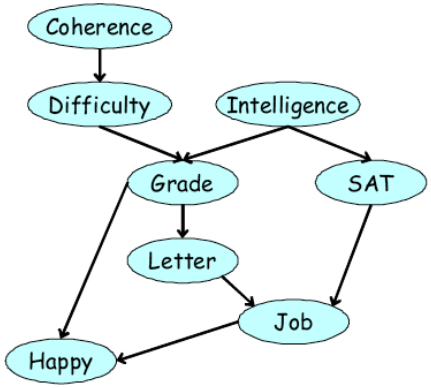
$$p(\mathbf{Y}|\mathbf{E}=\mathbf{e}) = \frac{p(\mathbf{Y},\mathbf{e})}{p(\mathbf{e})}$$

- Apply variable elimination algorithm to the task of computing $P(\mathbf{Y},\mathbf{e})$
- Replace each factor $\phi \in \Phi$ that has $\mathbf{E} \cap \mathrm{Scope}[\phi] \neq \emptyset$ with

$$\phi'(\mathbf{x}_{\mathrm{Scope}[\phi]-\mathbf{E}}) = \phi(\mathbf{x}_{\mathrm{Scope}[\phi]-\mathbf{E}}, \mathbf{e}_{\mathbf{E}\cap\mathrm{Scope}[\phi]})$$

- Then, eliminate the variables in $\mathcal{X} - \mathbf{Y} - \mathbf{E}$. The returned factor $\phi^*(\mathbf{Y})$ is $p(\mathbf{Y},\mathbf{e})$
- To obtain the conditional $p(\mathbf{Y} \mid \mathbf{e})$, normalize the resulting product of factors – the normalization constant is $p(\mathbf{e})$

Running time of variable elimination

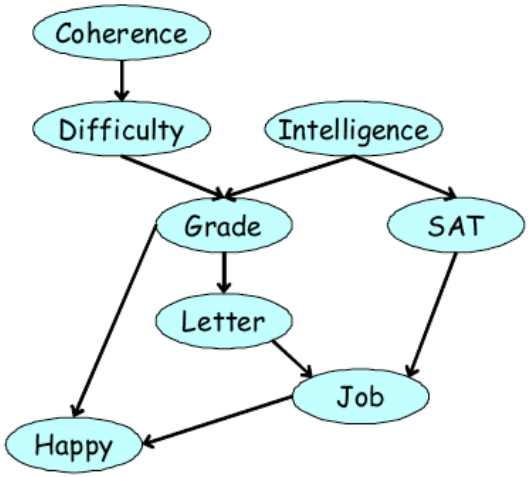


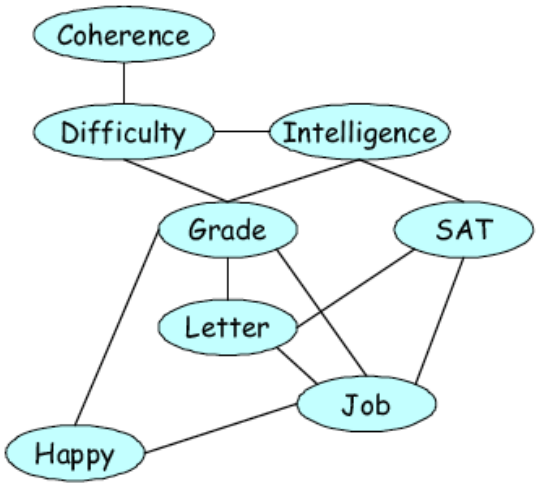
Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$

- Let n be the number of variables, and m the number of initial factors
- At each step, we pick a variable X_i and multiply all factors involving X_i, resulting in a single factor ψ_i
- Let N_i be the number of variables in the factor ψ_i, and let $N_{max} = \max_i N_i$
- The running time of VE is then $O(mk^{N_{max}})$, where $k = |\mathrm{Val}(X)|$. Why?
- The primary concern is that N_{max} can potentially be as large as n

Running time in graph-theoretic concepts

- Let's try to analyze the complexity in terms of the graph structure
- G_Φ is the undirected graph with one node per variable, where there is an edge (X_i, X_j) if these appear together in the scope of some factor ϕ
- Ignoring evidence, this is either the original MRF (for sum-product VE on MRFs) or the moralized Bayesian network:





Elimination as graph transformation

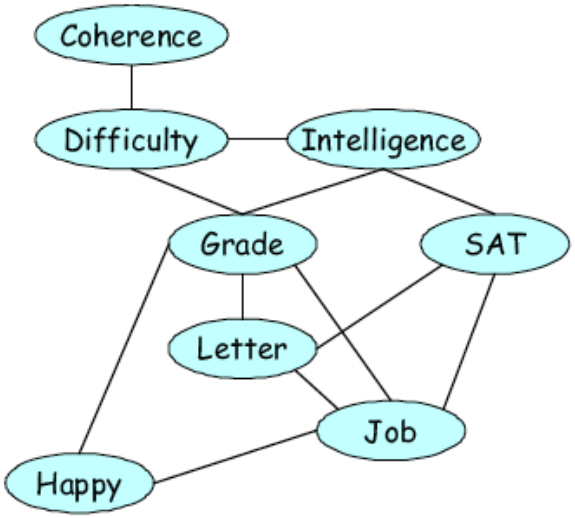
When a variable X is eliminated,

- We create a single factor ψ that contains X and all of the variables $\mathbf{Y}$ with which it appears in factors
- We eliminate X from ψ, replacing it with a new factor τ that contains all of the variables $\mathbf{Y}$, but not X. Let's call the new set of factors Φ_X

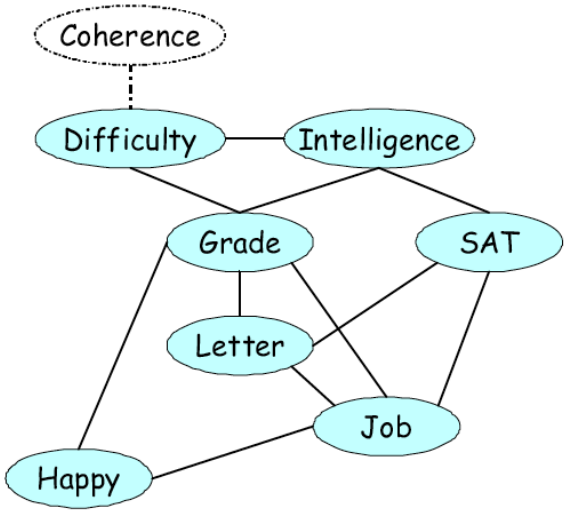
How does this modify the graph, going from G_Φ to G_{Φ_X}?

- Constructing ψ generates edges between all of the variables $Y \in \mathbf{Y}$
- Some of these edges were already in G_Φ, some are new
- The new edges are called **fill edges**
- The step of removing X from Φ to construct Φ_X removes X and all its incident edges from the graph

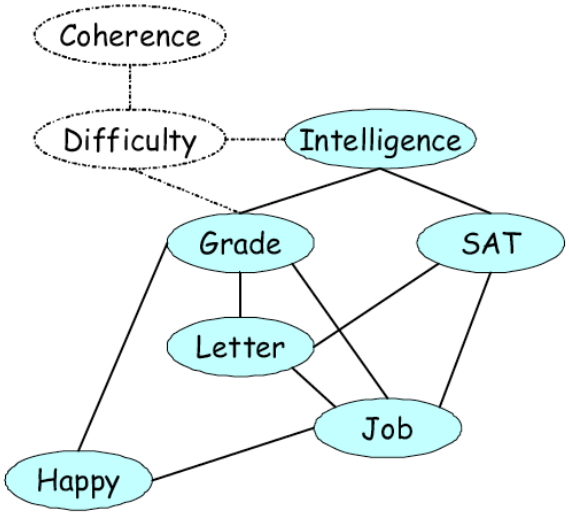
Example



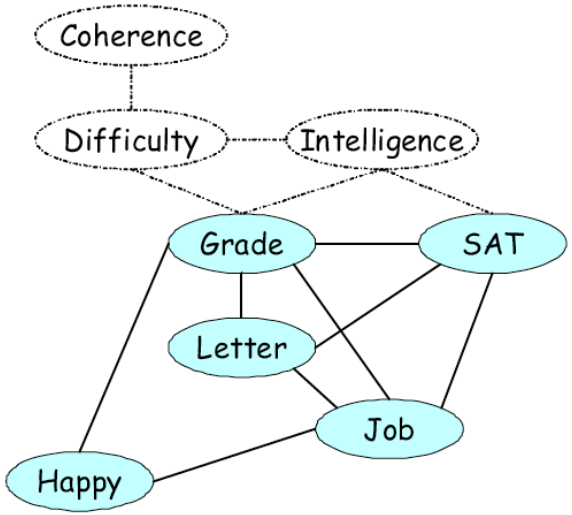
(Graph)



(Elim. C)



(Elim. D)

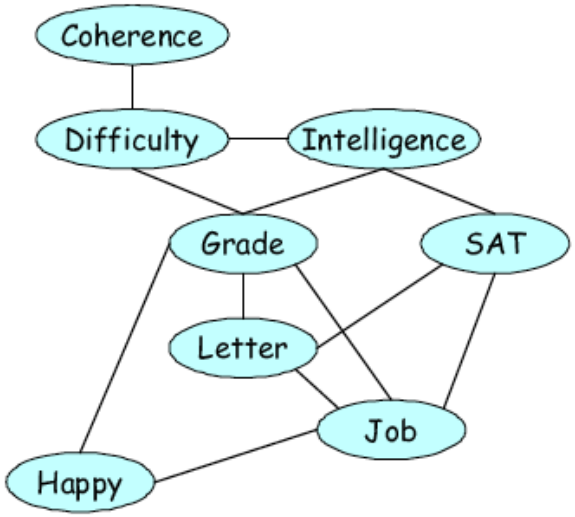


(Elim. I)

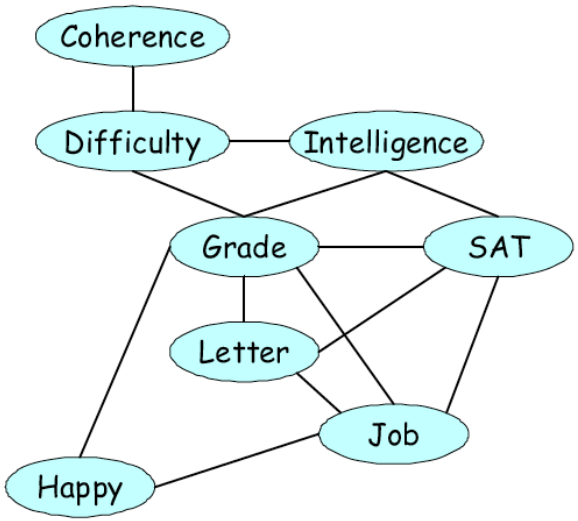
Induced graph

- We can summarize the computation cost using a single graph that is the union of all the graphs resulting from each step of the elimination
- We call this the **induced graph** $\mathcal{I}_{\Phi,\prec}$, where $\prec$ is the elimination ordering

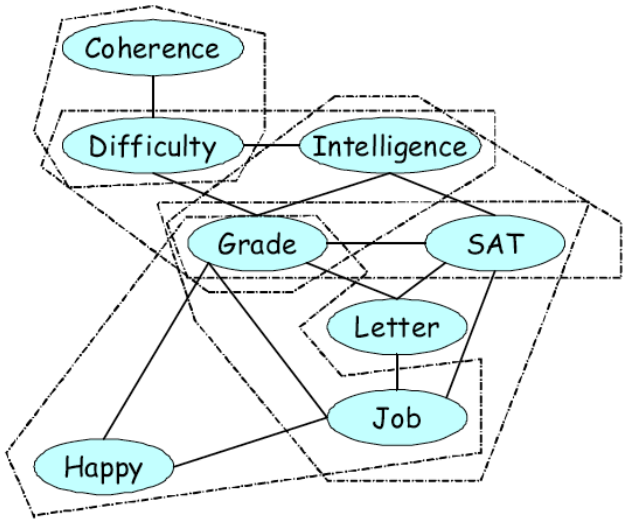
Example



Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$



(Induced graph)



(Maximal Cliques)

Properties of the induced graph

- **Theorem**: Let $\mathcal{I}_{\Phi,\prec}$ be the induced graph for a set of factors Φ and ordering $\prec$, then
  1. Every factor generated during VE has a scope that is a clique in $\mathcal{I}_{\Phi,\prec}$
  2. Every maximal clique in $\mathcal{I}_{\Phi,\prec}$ is the scope of some intermediate factor in the computation

  (see Koller & Friedman for proof)
- Thus, N_{max} is equal to the size of the largest clique in $\mathcal{I}_{\Phi,\prec}$
- The running time, $O(mk^{N_{max}})$, is exponential in the size of the largest clique of the induced graph

Induced width

- The **width** of an induced graph is #nodes in largest clique - 1
- We define the **induced width** $w_{\mathcal{G},\prec}$ to be the width of the graph $\mathcal{I}_{\mathcal{G},\prec}$ induced by applying VE to $\mathcal{G}$ using ordering $\prec$
- The **treewidth**, or "minimal induced width" of graph $\mathcal{G}$ is

$$w^*_{\mathcal{G}} = \min_{\prec} w_{\mathcal{G},\prec}$$

- The treewidth provides a bound on the best running time achievable by VE on a distribution that factorizes over $\mathcal{G}$: $O(mk^{w^*_{\mathcal{G}}+1})$,
- Unfortunately, finding the **best** elimination ordering (equivalently, computing the treewidth) for a graph is NP-hard
- In practice, heuristics are used to find a good elimination ordering

Choosing an elimination ordering

Set of possible heuristics:

- **Min-fill**: the cost of a vertex is the number of edges that need to be added to the graph due to its elimination.
- **Weighted-Min-Fill**: the cost of a vertex is the sum of weights of the edges that need to be added to the graph due to its elimination. Weight of an edge is the product of weights of its constituent vertices.
- **Min-neighbors**: The cost of a vertex is the number of neighbors it has in the current graph.
- **Min-weight**: the cost of a vertex is the product of weights (domain cardinality) of its neighbors.

Which one better?

- None of these criteria is better than others.
- Often will try several.

Today's lecture

- Exact inference
  1. Worst-case complexity of probabilistic inference
  2. Elimination algorithm
  3. Running-time analysis of elimination algorithm (*treewidth*)
- Approximate inference

Approximate marginal inference

- Given the joint $p(x_1, \ldots, x_n)$ represented as a graphical model, how do we perform **marginal inference**, e.g. to compute $p(x_1 \mid e)$?
- We showed earlier in the lecture that doing this exactly is NP-hard
- Nearly all *approximate inference* algorithms are either:
  1. Monte-carlo methods (e.g., **Gibbs sampling**, likelihood reweighting, MCMC)
  2. Variational algorithms (e.g., mean-field, loopy belief propagation)

Generating samples from a Bayesian network

Algorithm 12.1 Forward Sampling in a Bayesian network

Procedure Forward-Sample (
  $\mathcal{B}$ // Bayesian network over $\mathcal{X}$
)

1 Let $X_1, \ldots, X_n$ be a topological ordering of $\mathcal{X}$
2 **for** $i = 1, \ldots, n$
3 $\boldsymbol{u}_i \leftarrow \boldsymbol{x}\langle \mathrm{Pa}_{X_i} \rangle$ // Assignment to Pa_{X_i} in $x_1, \ldots, x_{i-1}$
4 Sample x_i from $P(X_i \mid \boldsymbol{u}_i)$
5 **return** $(x_1, \ldots, x_n)$

(Koller & Friedman, *Probabilistic Graphical Models*, MIT Press 2009)

Monte-Carlo algorithms

- Given a joint distribution $p(x_1, \ldots, x_n)$, how do we compute marginals?

$$\begin{aligned} p[X_1 = x_1] &= E_{\mathbf{x}\sim p}[f(\mathbf{x})], \text{ where } f(\mathbf{x}) = 1[X_1 = x_1] \\ &= \sum_{\mathbf{x}} p(\mathbf{x}) f(\mathbf{x}). \end{aligned}$$

- Rather than explicitly enumerating *all* assignments, consider the following Monte-Carlo estimate of the expectation:

$$\begin{aligned} \mathbf{x}^1 &\sim p(\mathbf{x}) \\ \mathbf{x}^2 &\sim p(\mathbf{x}) \\ &\vdots \\ \mathbf{x}^M &\sim p(\mathbf{x}) \end{aligned}$$

- Then, our *estimate* is $\hat{E}_p[f(x)] = \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}^m)$. How good is it?

Monte-Carlo algorithms

- Let $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$. Since $\mathcal{D}$ was drawn randomly from $p(\mathbf{x})$, the estimate is itself a random variable
- The estimate is *unbiased* because

$$\begin{aligned} E_{\mathbf{x}^1,\ldots,\mathbf{x}^M \sim p(\mathbf{x})}\left[\hat{E}[f(\mathbf{x})]\right] &= E_{\mathbf{x}^1,\ldots,\mathbf{x}^M \sim p(\mathbf{x})}\left[\frac{1}{M}\sum_{m=1}^{M} f(\mathbf{x}^m)\right] \\ &= \frac{1}{M}\sum_{m=1}^{M} E_{\mathbf{x}^m \sim p(\mathbf{x})}\left[f(\mathbf{x}^m)\right] \\ &= E_{\mathbf{x}\sim p(\mathbf{x})}[f(\mathbf{x})]. \end{aligned}$$

- How quickly does the estimate converge to the true expectation?

Law of large numbers

- There are two general results we can use, depending on whether we care about additive or multiplicative error
- **Hoeffding bound** says that:

$$\Pr_{\mathcal{D}\sim p(\mathbf{x})}\left[E_p[f(\mathbf{x})] - \epsilon \leq \hat{E}_{\mathcal{D}}[f(\mathbf{x})] \leq E_p[f(\mathbf{x})] + \epsilon\right] \geq 1 - 2e^{-2M\epsilon^2}$$

- **Chernoff bound** says that (assuming $f(\mathbf{x}) \in [0,1]$):

$$\Pr_{\mathcal{D}\sim p(\mathbf{x})}\left[E_p[f(\mathbf{x})](1-\epsilon) \leq \hat{E}_{\mathcal{D}}[f(\mathbf{x})] \leq E_p[f(\mathbf{x})](1+\epsilon)\right] \geq 1 - 2e^{\frac{-M\epsilon^2}{3}E_p[f(\mathbf{x})]}$$

- Estimating *single-variable* marginals for a BN is easy: just forward sample!
- What about computing *conditional* queries such as $p(\mathbf{X} = \mathbf{x} \mid \mathbf{E} = \mathbf{e})$?
- Computing denominator of $p(\mathbf{X} = \mathbf{x}, \mathbf{E} = \mathbf{e})/p(\mathbf{E} = \mathbf{e})$ needs $\Omega(1/p(\mathbf{E} = \mathbf{e}))$ samples, by Chernoff bound.

Monte-Carlo algorithms

- If we could instead directly sample from $p(\mathbf{X} \mid \mathbf{E} = \mathbf{e})$, we would be in business – but this is hard!
- For the same reason, sampling from an undirected graphical model $p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c)$ – even without evidence – is hard
- *Gibbs sampling* is an iterative algorithm that is guaranteed to eventually provide a sample from $p(\mathbf{X} \mid \mathbf{E} = \mathbf{e})$
  - Applies to both Bayesian networks and MRFs
  - First we will introduce it as a tool (that you will apply in PS4)
  - Lecture 6 (Oct. 24th) will show that Gibbs sampling is a special case of *Markov chain Monte-Carlo* (MCMC), and discuss the theory of why it works

Gibbs Sampling

- The GS algorithm:
  1. Suppose the graphical model contains variables $x_1,\ldots,x_n$
  2. Initialize starting values for $x_1,\ldots,x_n$
  3. Do until convergence:
     1. Pick an ordering of the n variables (can be fixed or random)
     2. For each variable x_i in order:
        1. Sample $x \sim P(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, i.e. the conditional distribution of x_i given the current values of all other variables
        2. Update $x_i \leftarrow x$

- When we update x_i, we <u>immediately</u> use its new value for sampling other variables x_j

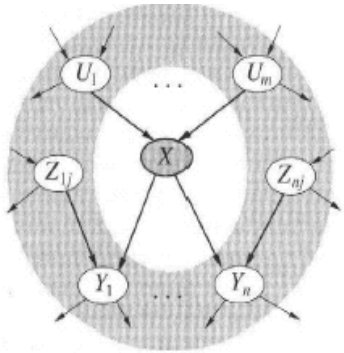
Slides adapted from Eric Xing (CMU)

Markov Blankets

- The conditional $P(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ looks intimidating, but recall Markov Blankets:
  - Let $MB(x_i)$ be the Markov Blanket of x_i, then

$$P(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i \mid MB(x_i))$$

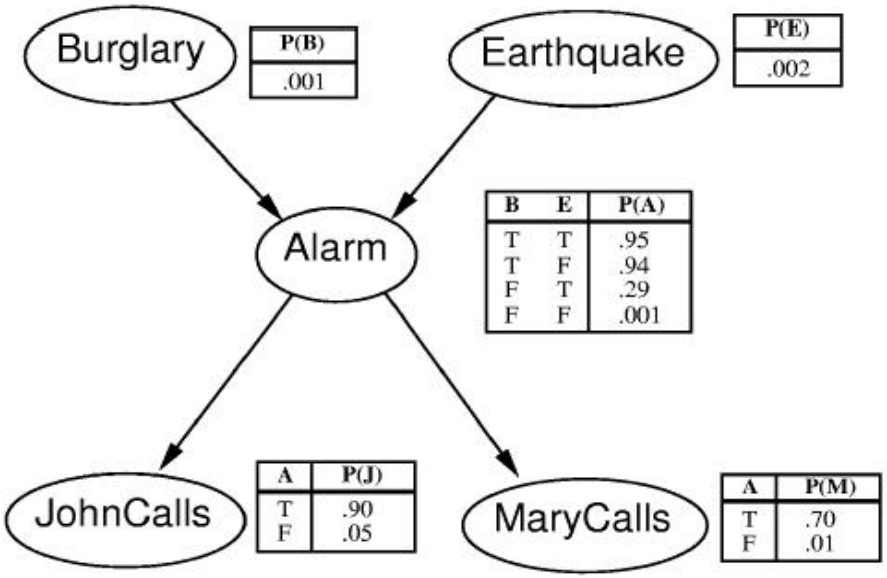
- For a BN, the Markov Blanket of x_i is the set containing its parents, children, and co-parents



- For an MRF, the Markov Blanket of x_i is its immediate neighbors

Slides adapted from Eric Xing (CMU)

Gibbs Sampling: An Example

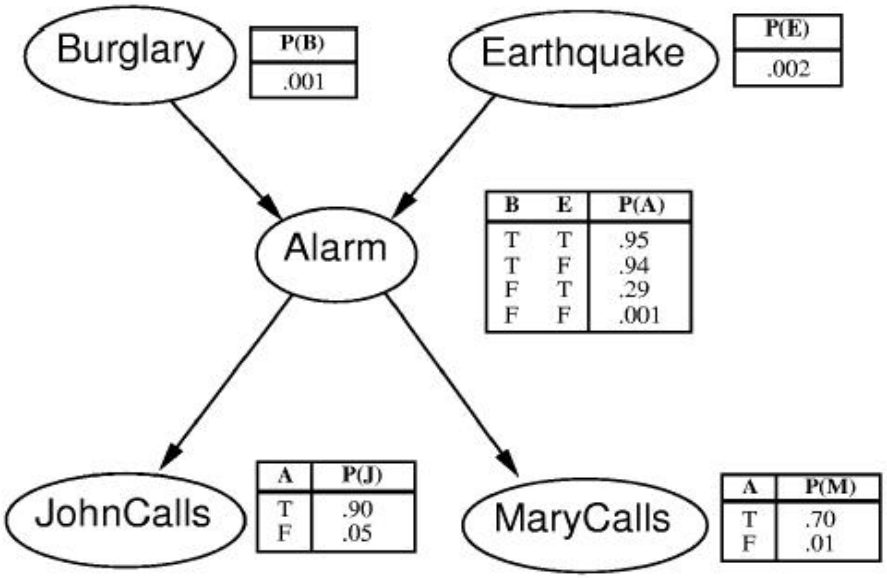


t	B	E	A	J	M
0	F	F	F	F	F
1					
2					
3					
4					

- Consider the alarm network
  - Assume we sample variables in the order B,E,A,J,M
  - Initialize all variables at t = 0 to False

Slides adapted from Eric Xing (CMU)

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F				
2					
3					
4					

- Sampling P(B|A,E) at t = 1: Using Bayes Rule,

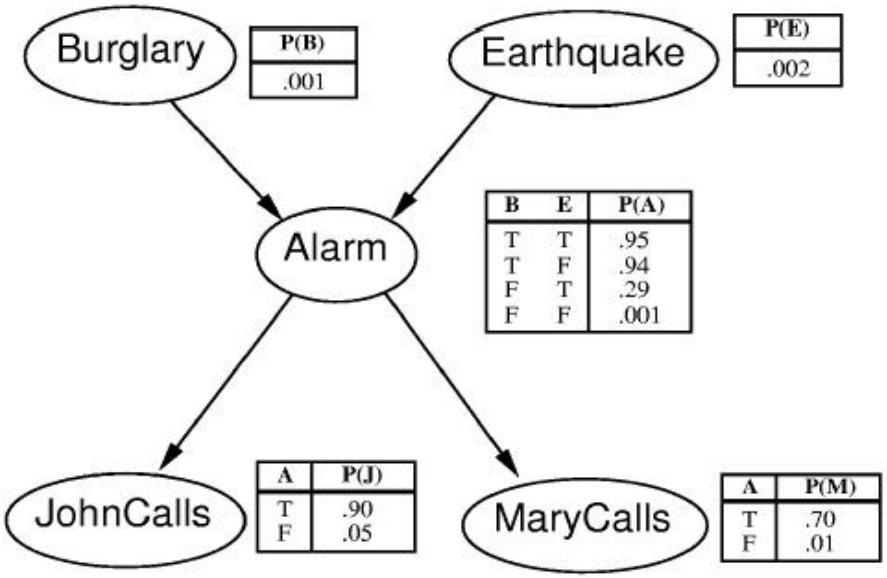
$$P(B \mid A, E) \quad P(A \mid B, E)P(B)$$

- A=false, E=false, so we compute:

$$P(B = T \mid A = F, E = F) \quad (0.06)(0.01) = 0.0006$$

$$P(B = F \mid A = F, E = F) \quad (0.999)(0.999) = 0.9980$$

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T			
2					
3					
4					

- Sampling P(E|A,B): Using Bayes Rule,

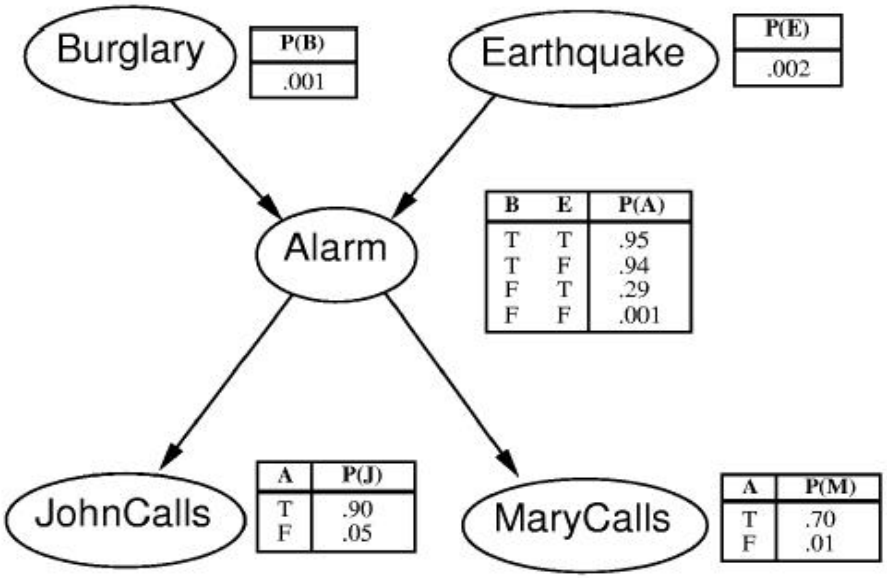
$$P(E \mid A, B) \quad P(A \mid B, E)P(E)$$

- (A,B) = (F,F), so we compute the following,

$$P(E = T \mid A = F, B = F) \quad (0.71)(0.02) = 0.0142$$

$$P(E = F \mid A = F, B = F) \quad (0.999)(0.998) = 0.9970$$

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F		
2					
3					
4					

- Sampling P(A|B,E,J,M): Using Bayes Rule,

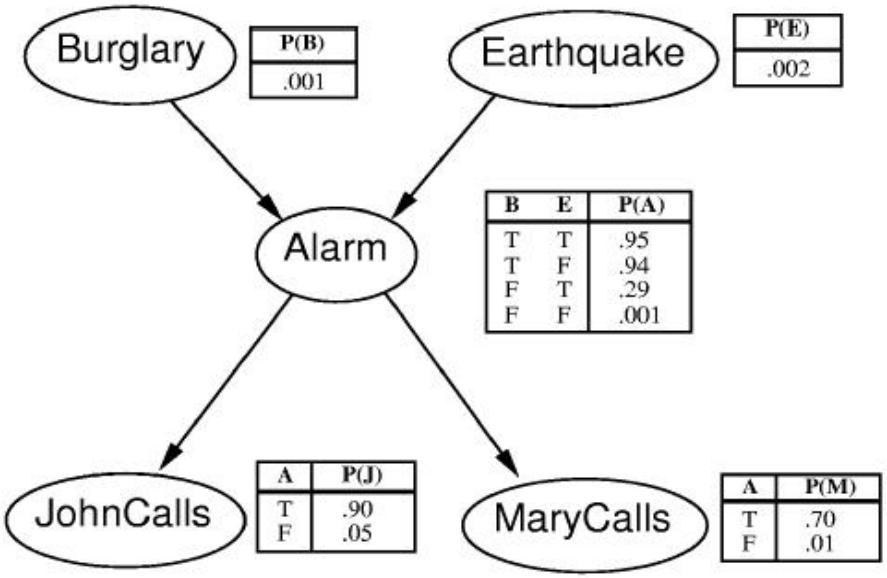
$$P(A \mid B, E, J, M) \quad P(J \mid A)P(M \mid A)P(A \mid B, E)$$

- (B,E,J,M) = (F,T,F,F), so we compute:

$$P(A = T \mid B = F, E = T, J = F, M = F) \quad (0.1)(0.3)(0.29) = 0.0087$$

$$P(A = F \mid B = F, E = T, J = F, M = F) \quad (0.95)(0.99)(0.71) = 0.6678$$

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	
2					
3					
4					

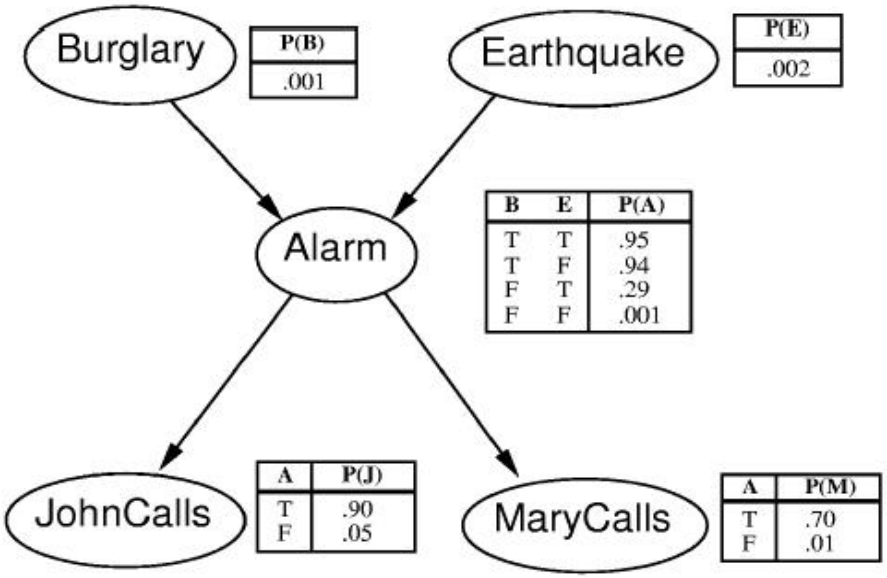
- Sampling P(J|A): No need to apply Bayes Rule

- A = F, so we compute the following, and sample

$$P(J = T \mid A = F) \quad 0.05$$

$$P(J = F \mid A = F) \quad 0.95$$

Gibbs Sampling: An Example



t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	F
2					
3					
4					

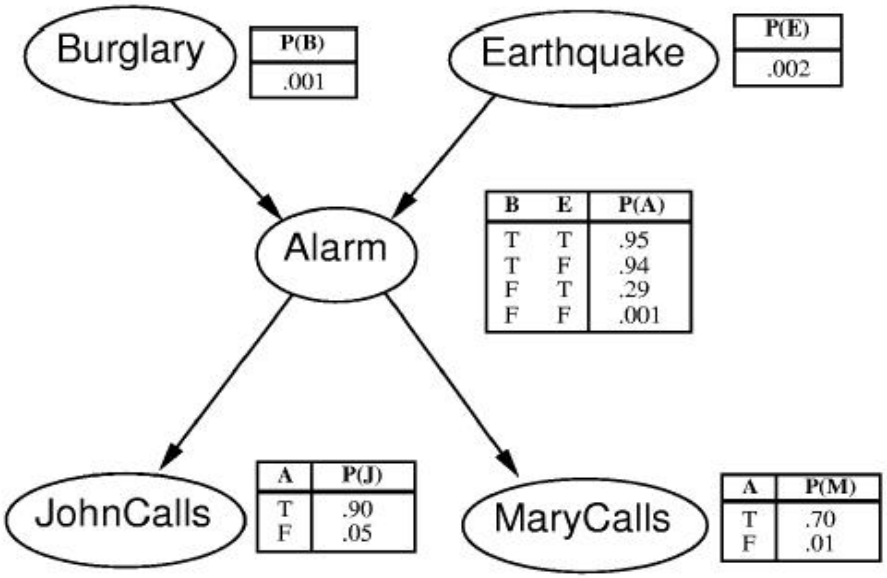
- Sampling P(M|A): No need to apply Bayes Rule

- A = F, so we compute the following, and sample

$$P(M = T \mid A = F) \quad 0.01$$

$$P(M = F \mid A = F) \quad 0.99$$

Gibbs Sampling: An Example

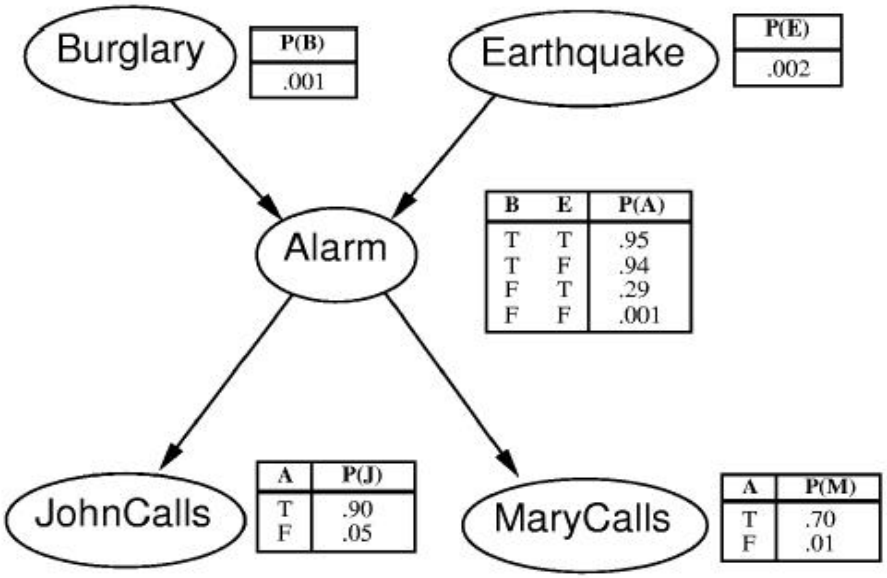


t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	F
2	F	T	T	T	T
3					
4					

- Now t = 2, and we repeat the procedure to sample new values of B,E,A,J,M …

Slides adapted from Eric Xing (CMU)

Gibbs Sampling: An Example



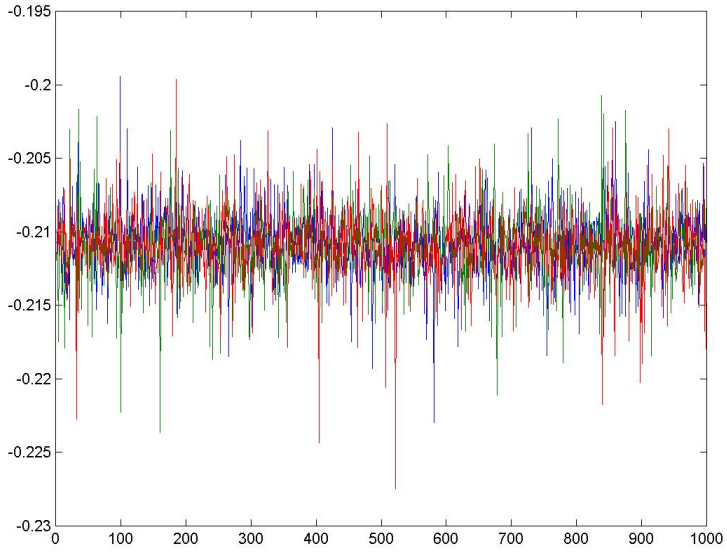
t	B	E	A	J	M
0	F	F	F	F	F
1	F	T	F	T	F
2	F	T	T	T	T
3	T	F	T	F	T
4	T	F	T	F	F

- Now t = 2, and we repeat the procedure to sample new values of B,E,A,J,M …
- And similarly for t = 3, 4, etc.

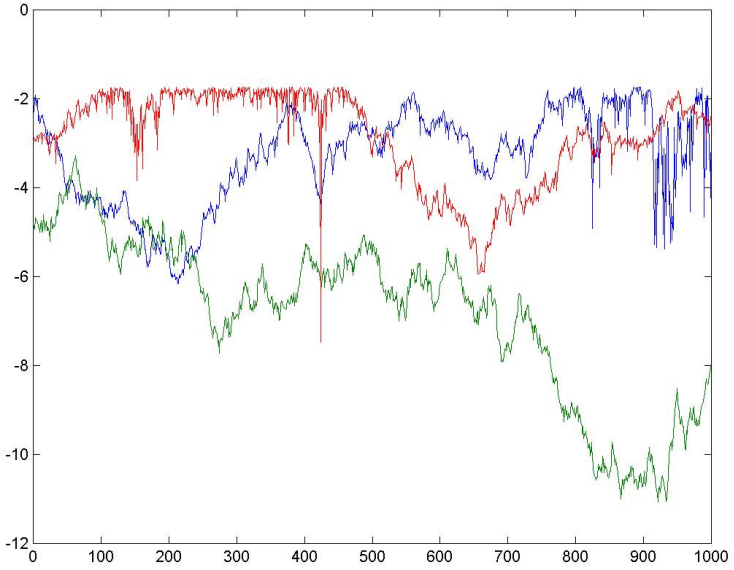
Slides adapted from Eric Xing (CMU)

Sample Values vs Time

Well-mixed chains



Poorly-mixed chains



- Monitor convergence by plotting samples (of variables) from multiple runs (”chains”) with different random seeds
  - If the chains are well-mixed (left), they are probably converged
  - If the chains are poorly-mixed (right), we should continue burn-in

Slides adapted from Eric Xing (CMU)