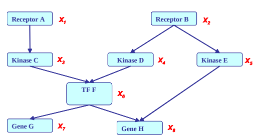
Machine Learning

10-701/15-781, Fall 2011

Bayesian Networks and Exact Inference

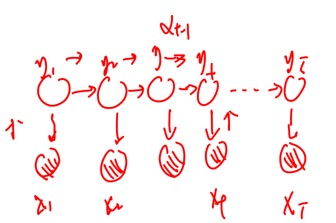
Eric Xing



Lecture 13-14, October 24, 2011

Reading: Chap. 8, C.B book

Recall HMM



(1) $P(x_1, x_2 \cdots x_T ; y_1 \cdots y_T)$

$= P(y_1) P(x_1|y_1) P(y_2|y_1) P(x_2|y_2)$ ....

$P(x_T | y_T, x_1 \cdots x_{T-1}, y_1 \cdots y_{T-1})$

$P(y_T | \cdots)$

(2) $P(y_1) \to P(x_T|y_T)$

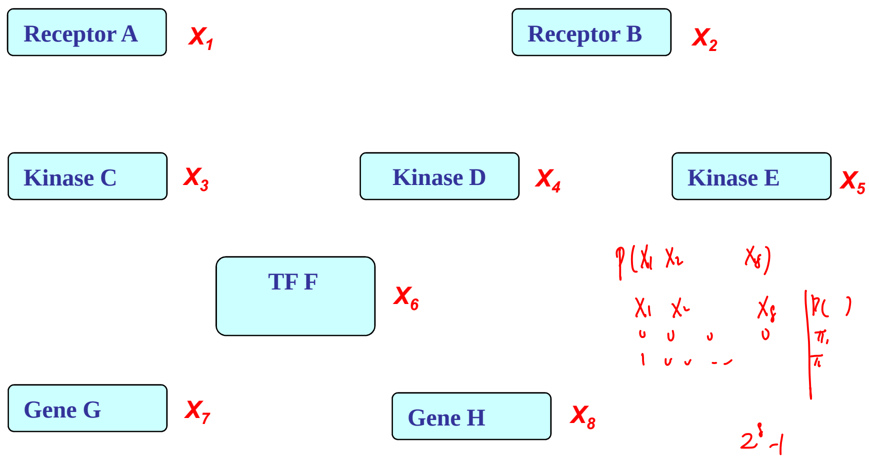
(3) α. β. V.

$\alpha_t^k = P(x_t|y_t) \sum_{k'} a_{kk'} \alpha_{t-1}(k')$

What is a Bayesian Network?

--- example from a signal transduction pathway

- A possible world for cellular signal transduction:

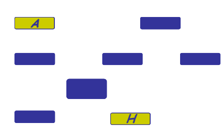


Recap of Basic Prob. Concepts

- **Representation: what is the joint probability dist. on multiple variables?**

$$P(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8,)$$

  - **How many state configurations in total? --- 2^8**
  - **Are they all needed to be represented?**
  - **Do we get any scientific/medical insight?**



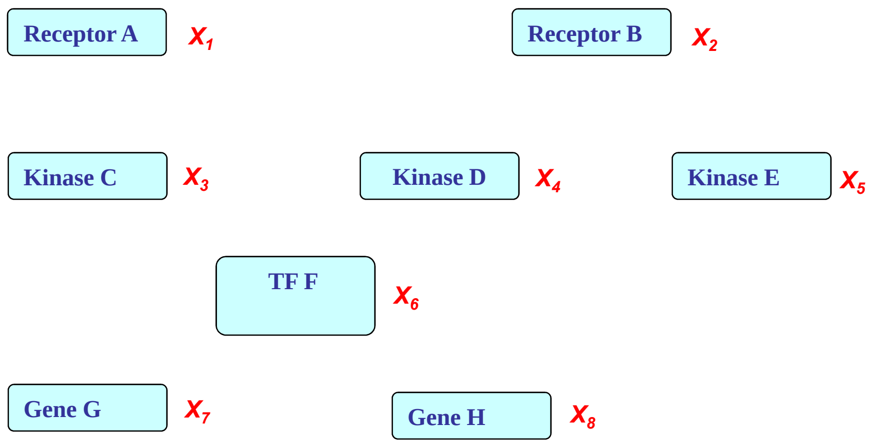
- **Learning: where do we get all this probabilities?**
  - **Maximal-likelihood estimation? but how many data do we need?**
  - **Where do we put domain knowledge in terms of plausible relationships between variables, and plausible values of the probabilities?**

- **Inference: If not all variables are observable, how to compute the conditional distribution of latent variables given evidence?**
  - **Computing *p(H|A)* would require summing over all 2^6 configurations of the unobserved variables**

What is a Bayesian Network?

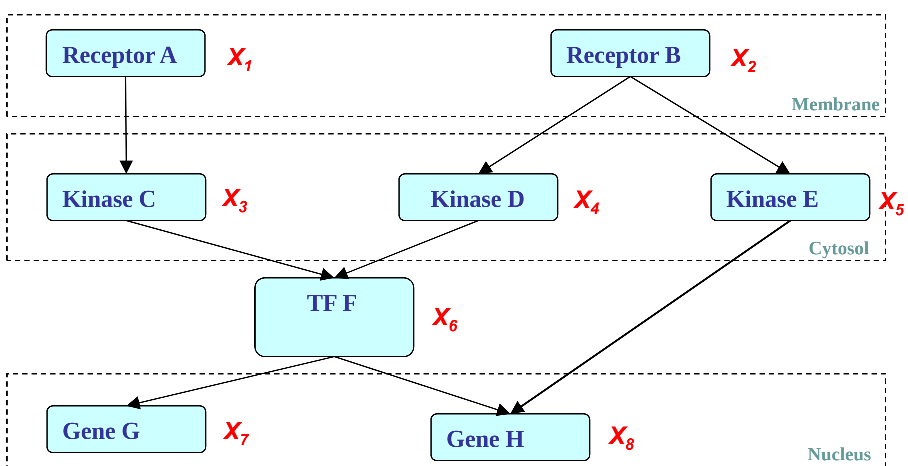
--- example from a signal transduction pathway

- A possible world for cellular signal transduction:



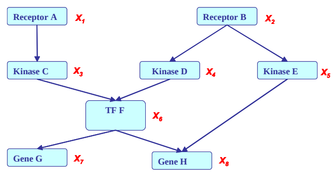
BN: Structure Simplifies Representation

- Dependencies among variables



Bayesian Network

- If X_i's are conditionally independent (as described by a BN), the joint can be factored to a product of simpler terms, e.g.,

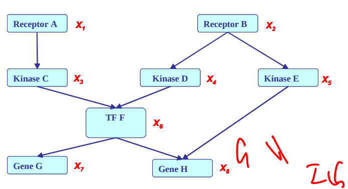


$$P(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8)$$
$$= P(X_1)\, P(X_2)\, P(X_3|\, X_1)\, P(X_4|\, X_2)\, P(X_5|\, X_2)\, P(X_6|\, X_3, X_4)\, P(X_7|\, X_6)\, P(X_8|\, X_5, X_6)$$

- Why we may favor a BN?
  - Representation cost: how many probability statements are needed?

    2+2+4+4+4+8+4+8=36, an 8-fold reduction from 2^8!
  - Algorithms for systematic and efficient inference/learning computation
    - Exploring the graph structure and probabilistic semantics
  - Incorporation of domain knowledge and causal (logical) structures

Bayesian Network: Factorization Theorem



$$P(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8)$$
$$= P(X_1)\, P(X_2)\, P(X_3|\, X_1)\, P(X_4|\, X_2)\, P(X_5|\, X_2)\, P(X_6|\, X_3, X_4)\, P(X_7|\, X_6)\, P(X_8|\, X_5, X_6)$$

- **Theorem:**

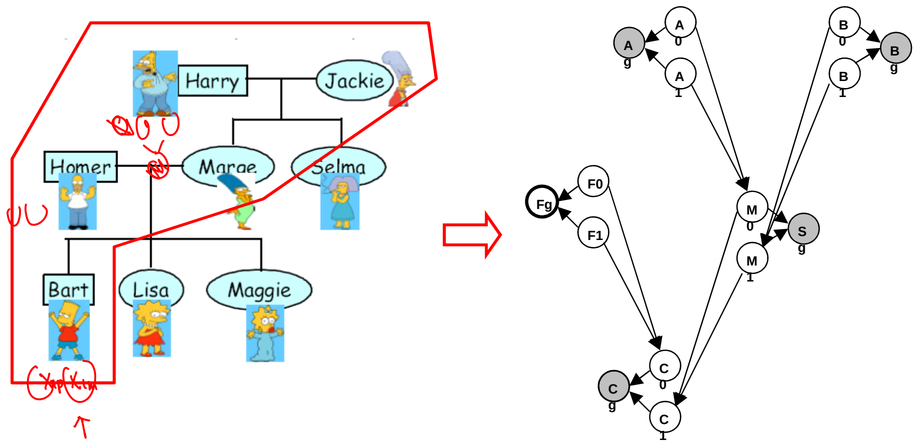
  Given a DAG, The most general form of the probability distribution that is consistent with the (probabilistic independence properties encoded in the) graph factors according to "node given its parents":

$$P(\mathbf{X}) = \prod_i P(X_i \mid \mathbf{X}_{\pi_i})$$

  where $\mathbf{X}_{\pi_i}$ is the set of parents of xi. d is the number of nodes (variables) in the graph.

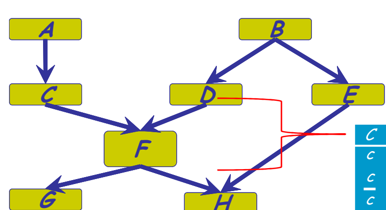
Example: a pedigree of people

- Genetic Pedigree



Specification of a BN

- There are two components to any GM:
  - the *qualitative* specification
  - the *quantitative* specification



C	D	P(F | C,D)	
c	d	0.9	0.1
c	$\bar{d}$	0.2	0.8
$\bar{c}$	d	0.9	0.1
$\bar{c}$	$\bar{d}$	0.01	0.99

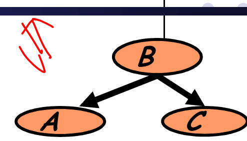
Qualitative Specification

- Where does the qualitative specification come from?
  - Prior knowledge of causal relationships
  - Prior knowledge of modular relationships
  - Assessment from experts
  - Learning from data
  - We simply link a certain architecture (e.g. a layered graph)
  - …

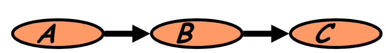
Local Structures & Independencies

$P(A \perp C \mid B) = P(A \mid B) P(C \mid B)$

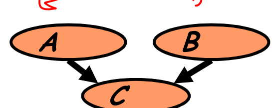
- Common parent
  - Fixing B decouples A and C
    "given the level of gene B, the levels of A and C are independent"



- Cascade
  - Knowing B decouples A and C
    "given the level of gene B, the level gene A provides no extra prediction value for the level of gene C"

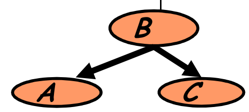


- V-structure
  - Knowing C couples A and B
    because A can "explain away" B w.r.t. C
    "If A correlates to C, then chance for B to also correlate to B will decrease"



- The language is compact, the concepts are rich!

A simple justification



$P(A,C|B) = P(A|B)\,P(C|B)$

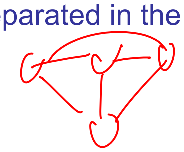
$\Uparrow$

$P(A,B,C) = P(B)\,P(A|B)\,P(C|B)$

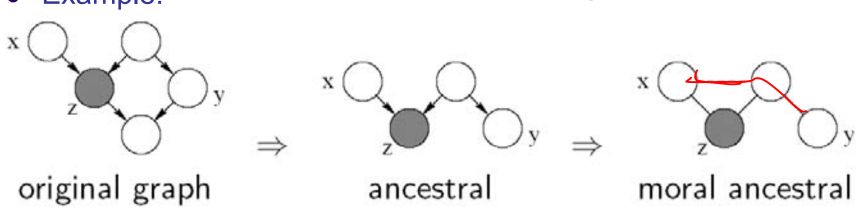
Graph separation criterion

- D-separation criterion for Bayesian networks (D for Directed edges):

  Definition: variables x and y are *D-separated* (conditionally independent) given z if they are separated in the *moralized* ancestral graph



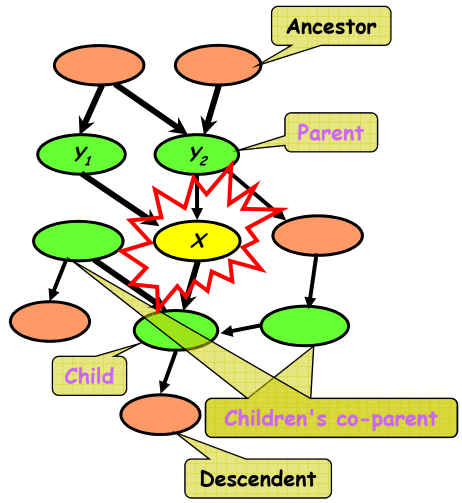
- Example:



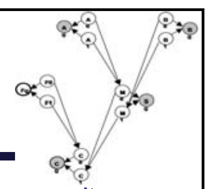
Local Markov properties of DAGs

Structure: *DAG*

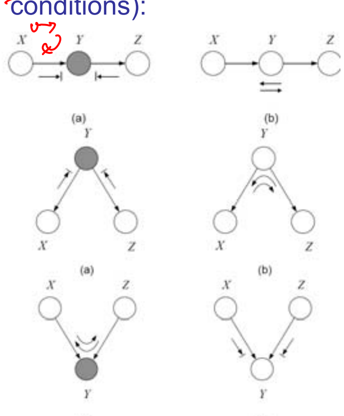
- **Meaning: a node is conditionally independent of every other node in the network outside its Markov blanket**
- **Local conditional distributions (CPD) and the DAG completely determine the joint dist.**
- **Give causality relationships, and facilitate a generative process**



Global Markov properties of DAGs



- X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "*Bayes-ball*" algorithm illustrated bellow (and plus some boundary conditions):

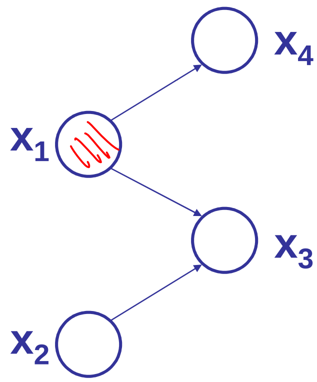


- **Defn: $I(G)$=all independence properties that correspond to d-separation:**

$$\mathrm{I}(G)=\left\{X \perp Z \middle| Y : \mathrm{dsep}_G(X;Z|Y)\right\}$$

- **D-separation is sound and complete**

Example:



- Complete the I(G) of this graph:

$X_2 \perp X_4 \mid \phi$

$X_4 \perp X_3 \mid X_1$

Essentially: A BN is a database of Pr. Independence statements among variables.

Towards quantitative specification of probability distribution

- Separation properties in the graph imply independence properties about the associated variables
- For the graph to be useful, any conditional independence properties we can derive from the graph should hold for the probability distribution that the graph represents
- **The Equivalence Theorem**

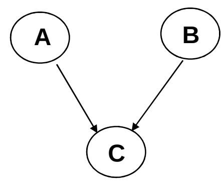
  For a graph G,

  Let $\mathcal{D}_1$ denote the family of all distributions that satisfy I(G),

  Let $\mathcal{D}_2$ denote the family of all distributions that factor according to G,

  Then $\mathcal{D}_1 \equiv \mathcal{D}_2$.

Quantitative Specification



parameterization

$\longleftrightarrow$ p(A,B,C) = P(A)·P(B)·P(C|A,B)

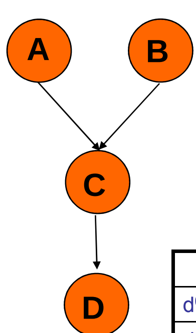
Conditional probability tables (CPTs)

a^0	0.75
a^1	0.25

b^0	0.33
b^1	0.67

I(G) **P(a,b,c.d) = P(a)P(b)P(c|a,b)P(d|c)**

P(A,B,C,D)



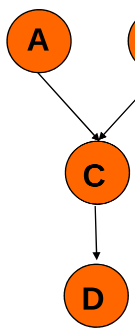
	a^0b^0	a^0b^1	a^1b^0	a^1b^1
c^0	0.45	1	0.9	0.7
c^1	0.55	0	0.1	0.3

	c^0	c^1
d^0	0.3	0.5
d^1	07	0.5

Conditional probability density func. (CPDs)

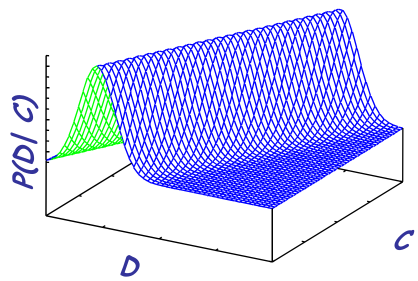
$A \sim N(\mu_a, \Sigma_a)$ $B \sim N(\mu_b, \Sigma_b)$

P(a,b,c.d) =
P(a)P(b)P(c|a,b)P(d|c)



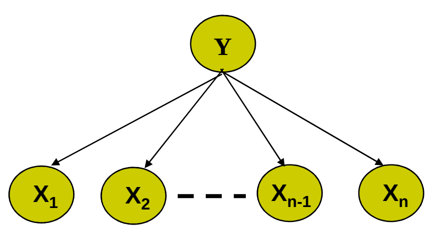
$C \sim N(A+B, \Sigma_c)$

$D \sim N(\mu_a+C, \Sigma_a)$



 21

Conditional Independencies



Label

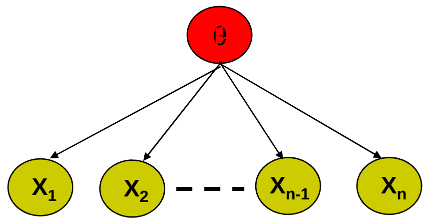
Features

What is this model

1. **When Y is observed?**
2. **When Y is unobserved?**

 22

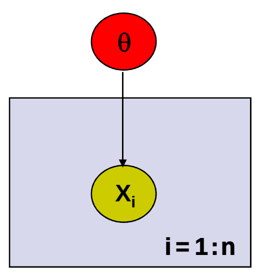
Conditionally Independent Observations



Model parameters

Data = $\{y_1, \ldots y_n\}$

"Plate" Notation



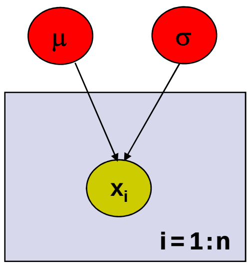
Model parameters

Data = $\{x_1, \ldots x_n\}$

Plate = rectangle in graphical model

variables within a plate are replicated
in a conditionally independent manner

Example: Gaussian Model



Generative model:

$p(x_1,\ldots,x_n \mid \mu, \sigma) = \prod p(x_i \mid \mu, \sigma)$

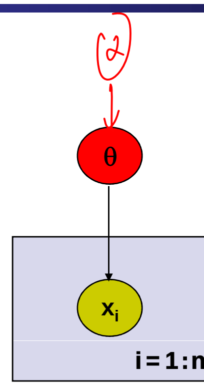
$= p(\text{data} \mid \text{parameters})$

$= p(D \mid \theta)$

where $\theta = \{\mu, \sigma\}$

- Likelihood = p(data | parameters)
  $= p(D \mid \theta)$
  $= L(\theta)$
- Likelihood tells us how likely the observed data are conditioned on a particular setting of the parameters
  - Often easier to work with log $L(\theta)$

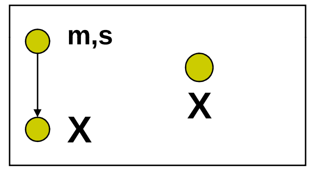
Bayesian models



More examples

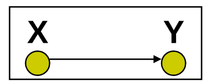
Density estimation

Parametric and nonparametric methods



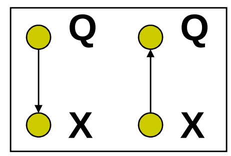
Regression

Linear, conditional mixture, nonparametric



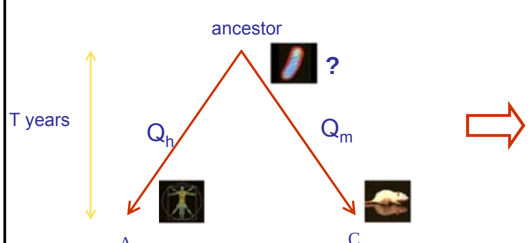
Classification

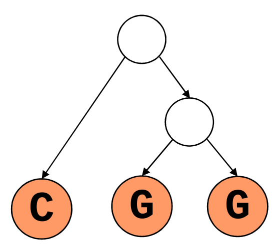
Generative and discriminative approach



Example, con'd

- Evolution

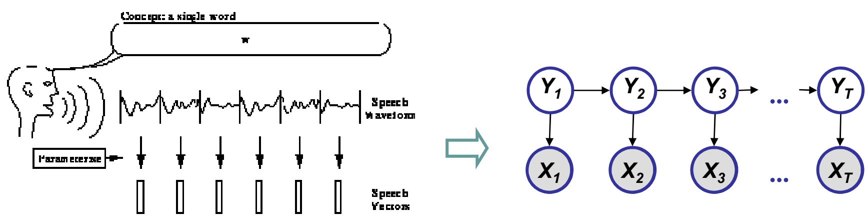




Tree Model

Example, con'd

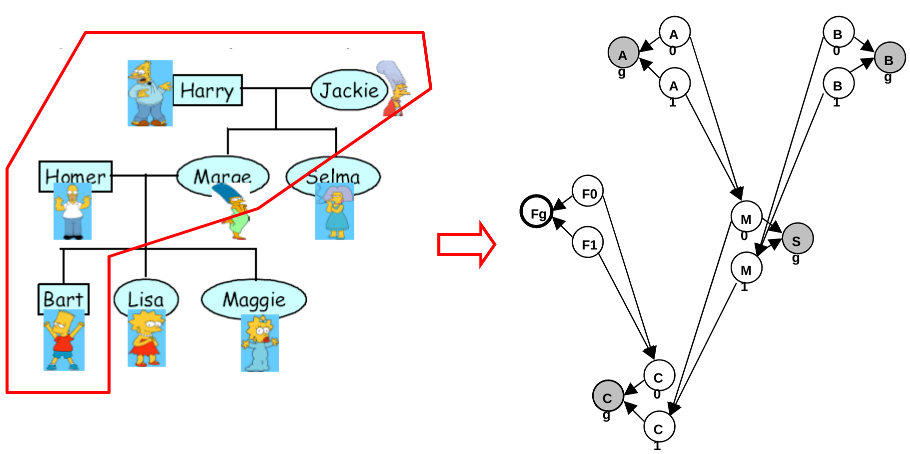
- Speech recognition



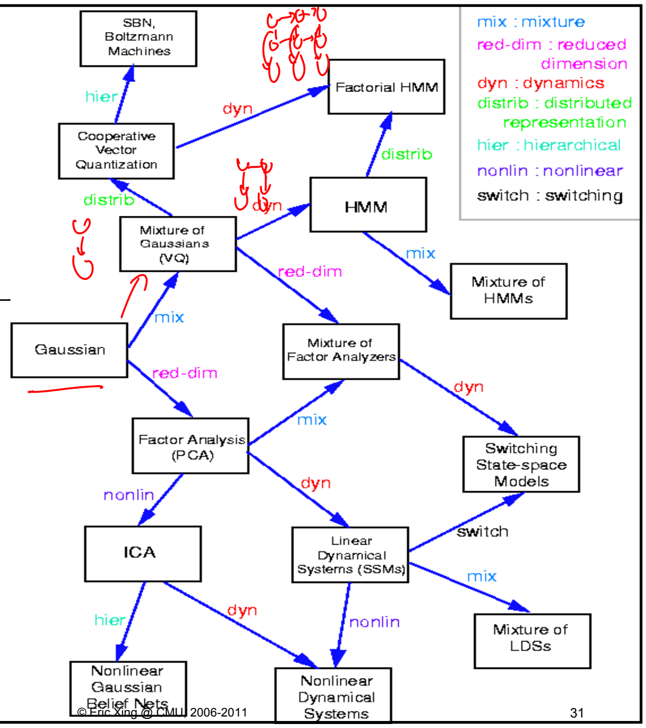
Hidden Markov Model

Example, con'd

- Genetic Pedigree



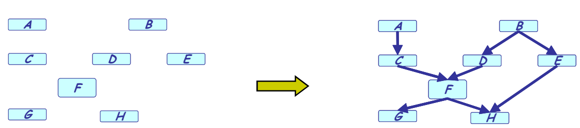
An (incomplete) genealogy of BNs



(Picture by Zoubin Ghahramani and Sam Roweis)

BN and Graphical Models

- A Bayesian network is a special case of **Graphical Models**
- A Graphical Model refers to a family of distributions on a set of random variables that are **compatible** with all the probabilistic independence propositions encoded by a graph that connects these variables
- It is a smart way to write/specify/compose/design exponentially-large probability distributions without paying an exponential cost, and at the same time endow the distributions with structured semantics



$P(X_1,X_2,X_3,X_4,X_5,X_6,X_7,X_8)$

$P(X_{1:8}) = P(X_1)P(X_2)P(X_3 \mid X_1,X_2)P(X_4 \mid X_2)P(X_5 \mid X_2)$
$P(X_6 \mid X_3,X_4)P(X_7 \mid X_6)P(X_8 \mid X_5,X_6)$

Summary

- Represent dependency structure with a directed acyclic graph
  - Node <-> random variable
  - Edges encode dependencies
    - Absence of edge -> conditional independence
  - Plate representation
  - A BN is a database of prob. Independence statement on variables

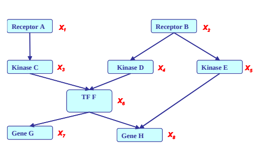
- The factorization theorem of the joint probability
  - Local specification → globally consistent distribution
  - Local representation for exponentially complex state-space
- Support efficient inference and learning

Machine Learning

10-701/15-781, Fall 2011

Bayesian Networks and Exact Inference

Eric Xing



Lecture 13-14, October 24, 2011

Reading: Chap. 8, C.B book

Inference and Learning

- We now have compact representations of probability distributions: **BN**
- A BN *M* describes a unique probability distribution *P*
- Typical tasks:
  - Task 1: How do we answer **queries** about *P*?
    - We use **inference** as a name for the process of computing answers to such queries
  - Task 2: How do we estimate a **plausible model** *M* from data *D*?
    i. We use **learning** as a name for the process of obtaining point estimate of *M*.
    ii. But for *Bayesian*, they seek $p(M \mid D)$, which is actually an **inference** problem.
    iii. When not all variables are observable, even computing point estimate of *M* need to do **inference** to impute the *missing data*.

Probabilistic Inference

- **Computing statistical queries regarding the network, e.g.:**
  - **Is node X independent on node Y given nodes Z,W ?**
  - **What is the probability of X=true if (Y=false and Z=true)?**
  - **What is the joint distribution of (X,Y) if Z=false?**
  - **What is the likelihood of some full assignment?**
  - **What is the most likely assignment of values to all or a subset the nodes of the network?**
- **General purpose algorithms exist to fully automate such computation**
  - **Computational cost depends on the topology of the network**
  - **Exact inference:**
    - **The junction tree algorithm**
  - **Approximate inference;**
    - **Loopy belief propagation, variational inference, Monte Carlo sampling**

Inferential Query 1: Likelihood

- Most of the queries one may ask involve **evidence**
  - Evidence $\mathbf{x}_v$ is an assignment of values to a set $\mathbf{X}_v$ of nodes in the GM over varialbe set $\mathbf{X}=\{X_1, X_2, \ldots, X_n\}$
  - Without loss of generality $\mathbf{X}_v=\{X_{k+1}, \ldots , X_n\}$,
  - Write $\mathbf{X}_H=\mathbf{X}\backslash\mathbf{X}_v$ as the set of hidden variables, $\mathbf{X}_H$ can be $\varnothing$ or $\mathbf{X}$
- Simplest query: compute probability of evidence

$$P(\mathbf{x}_v) = \sum_{\mathbf{x}_H} P(\mathbf{X}_H, , \mathbf{X}_v) = \sum_{x_1} \cdots \sum_{x_k} P(x_1, \ldots, x_k, \mathbf{x}_v)$$

  - this is often referred to as computing the **likelihood** of $\mathbf{x}_v$

Inferential Query 2: Conditional Probability

- Often we are interested in the **conditional probability distribution** of a variable given the evidence

$$P(\mathbf{X}_H \mid \mathbf{X}_V = \mathbf{x}_V) = \frac{P(\mathbf{X}_H, \mathbf{x}_V)}{P(\mathbf{x}_V)} = \frac{P(\mathbf{X}_H, \mathbf{x}_V)}{\sum_{\mathbf{x}_H} P(\mathbf{X}_H = \mathbf{x}_H, \mathbf{x}_V)}$$

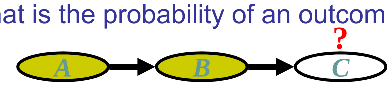
  - this is the ***a posteriori* belief** in $\mathbf{X}_H$, given evidence $\mathbf{x}_v$
- We usually query a subset $\mathbf{Y}$ of all hidden variables $\mathbf{X}_H=\{\mathbf{Y},\mathbf{Z}\}$ and "don't care" about the remaining, $\mathbf{Z}$:

$$P(\mathbf{Y} \mid \mathbf{x}_V) = \sum_{\mathbf{z}} P(\mathbf{Y}, \mathbf{Z} = \mathbf{z} \mid \mathbf{x}_V)$$

  - the process of summing out the "don't care" variables *z* is called **marginalization**, and the resulting $P(\mathbf{Y}|\mathbf{x}_v)$ is called a **marginal** prob.

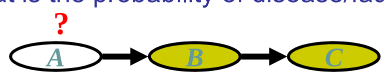
Applications of *a posteriori* Belief

- **Prediction**: what is the probability of an outcome given the starting condition



  - the query node is a descendent of the evidence

- **Diagnosis**: what is the probability of disease/fault given symptoms



  - the query node an ancestor of the evidence

- **Learning** under partial observation
  - fill in the unobserved values under an "EM" setting (more later)

- The directionality of information flow between variables is not restricted by the directionality of the edges in a GM
  - probabilistic inference can combine evidence form all parts of the network

Inferential Query 3: Most Probable Assignment

- In this query we want to find the **most probable joint assignment** (MPA) for ***some*** variables of interest

- Such reasoning is usually performed under some given evidence $\mathbf{x}_V$, and ignoring (the values of) other variables $\mathbf{Z}$:

$$\mathbf{Y}^* \mid \mathbf{x}_V = \arg\max_{\mathbf{y}} P(\mathbf{Y} \mid \mathbf{x}_V) = \arg\max_{\mathbf{y}} \sum_{\mathbf{z}} P(\mathbf{Y}, \mathbf{Z} = \mathbf{z} \mid \mathbf{x}_V)$$

  - this is the **maximum *a posteriori*** configuration of $\mathbf{Y}$.

Complexity of Inference

Thm:

Computing $P(X_H=x_H|\, x_v)$ in an arbitrary GM is NP-hard

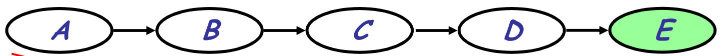
- **Hardness does not mean we cannot solve inference**
  - **It implies that we cannot find a general procedure that works efficiently for arbitrary GMs**
  - **For particular families of GMs, we can have provably efficient procedures**

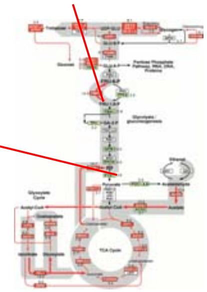
Approaches to inference

- Exact inference algorithms
  - The elimination algorithm
  - Belief propagation
  - The junction tree algorithms (but will not cover in detail here)
- Approximate inference techniques
  - Variational algorithms
  - Stochastic simulation / sampling methods
  - Markov chain Monte Carlo methods

Marginalization and Elimination

- A signal transduction pathway:





What is the likelihood that protein E is active?

- Query: $P(e)$

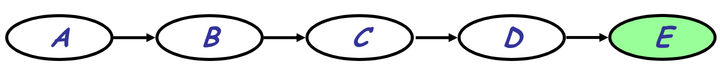
$$P(e) = \sum_d \sum_c \sum_b \sum_a P(a,b,c,d,e)$$

a naïve summation needs to enumerate over an exponential number of terms

- By chain decomposition, we get

$$= \sum_d \sum_c \sum_b \sum_a P(a)P(b \mid a)P(c \mid b)P(d \mid c)P(e \mid d)$$

Elimination on Chains

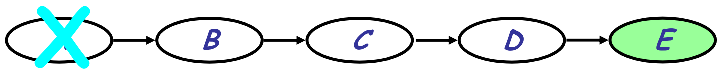


- Rearranging terms ...

$$P(e) = \sum_d \sum_c \sum_b \sum_a P(a)P(b \mid a)P(c \mid b)P(d \mid c)P(e \mid d)$$

$$= \sum_d \sum_c \sum_b P(c \mid b)P(d \mid c)P(e \mid d)\sum_a P(a)P(b \mid a)$$

Elimination on Chains

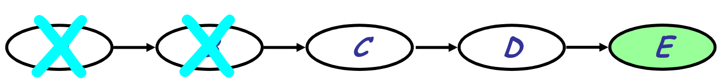


- Now we can perform innermost summation

$$P(e) = \sum_d \sum_c \sum_b P(c \mid b) P(d \mid c) P(e \mid d) \sum_a P(a) P(b \mid a)$$
$$= \sum_d \sum_c \sum_b P(c \mid b) P(d \mid c) P(e \mid d) p(b)$$

- This summation "eliminates" one variable from our summation argument at a "local cost".

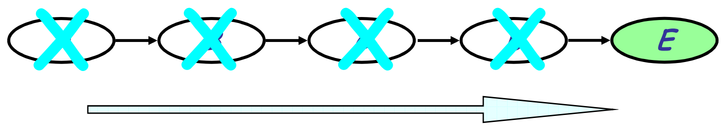
Elimination in Chains



- Rearranging and then summing again, we get

$$P(e) = \sum_d \sum_c \sum_b P(c \mid b) P(d \mid c) P(e \mid d) p(b)$$
$$= \sum_d \sum_c P(d \mid c) P(e \mid d) \sum_b P(c \mid b) p(b)$$
$$= \sum_d \sum_c P(d \mid c) P(e \mid d) p(c)$$

Elimination in Chains



- Eliminate nodes one by one all the way to the end, we get

$$P(e) = \sum_d P(e \mid d) p(d)$$

- Complexity:
  - Each step costs $O(|Val(X_i)|*|Val(X_{i+1})|)$ operations: $O(nk^2)$
  - Compare to naïve evaluation that sums over joint values of n-1 variables $O(k^n)$

Inference on General BN via Variable Elimination

General idea:

- Write query in the form

$$P(X_1, \boldsymbol{e}) = \sum_{x_n} \cdots \sum_{x_3} \sum_{x_2} \prod_i P(x_i \mid pa_i)$$

  - this suggests an "elimination order" of latent variables to be marginalized
- Iteratively
  - Move all irrelevant terms outside of innermost sum
  - Perform innermost sum, getting a new term
  - Insert the new term into the product
- wrap-up

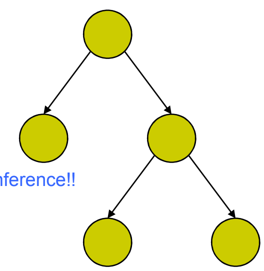
$$P(X_1 \mid \boldsymbol{e}) = \frac{P(X_1, \boldsymbol{e})}{P(\boldsymbol{e})}$$

From elimination to message passing

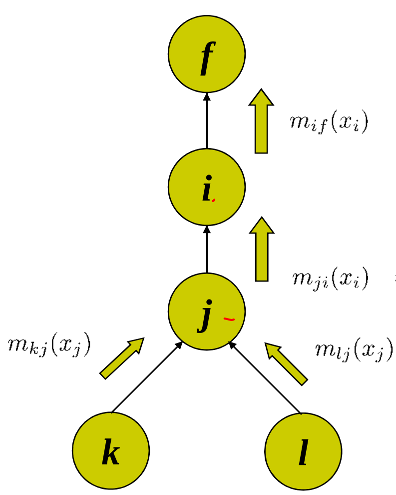
- Recall ELIMINATION algorithm:

  - Choose an ordering $\mathcal{Z}$ in which query node *f* is the final node
  - Place all potentials on an active list
  - Eliminate node *i* by removing all potentials containing *i*, take sum/product over x_i.
  - Place the resultant factor back on the list

- For a TREE graph:
  - Choose query node *f* as the root of the tree
  - View tree as a directed tree with edges pointing towards from *f*
  - Elimination ordering based on depth-first traversal
  - Elimination of each node can be considered as message-passing (or Belief Propagation) directly along tree branches, rather than on some transformed graphs
  - → thus, we can use the tree itself as a data-structure to do general inference!!



Message passing for trees



$$m_{ji}(x_i) = \sum_{x_j}\left(\psi(x_j)\psi(x_i,x_j)\prod_{k\in N(j)\setminus i} m_{kj}(x_j)\right)$$

Let $m_{ij}(x_i)$ denote the factor resulting from eliminating variables from bellow up to *i*, which is a function of x_i:

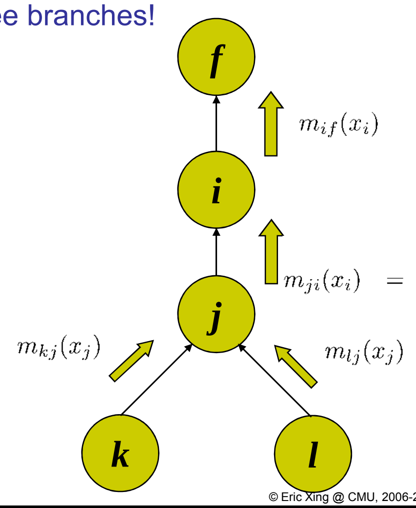
$$m_{ji}(x_i) = \sum_{x_j}\left(\psi(x_j)\psi(x_i,x_j)\prod_{k\in N(j)\setminus i} m_{kj}(x_j)\right)$$

This is reminiscent of a *message* sent from *j* to *i*.

$$p(x_f) \propto \psi(x_f)\prod_{e\in N(f)} m_{ef}(x_f)$$

$m_{ij}(x_i)$ represents a "belief" of x_i from x_j!

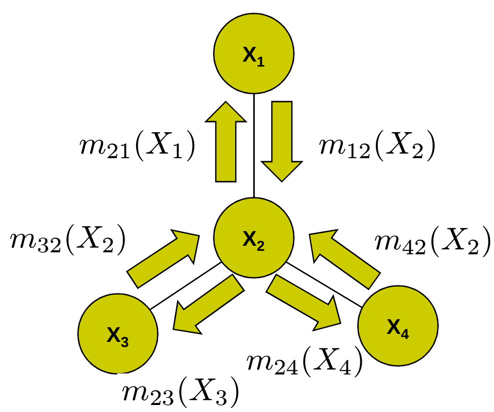
- Elimination on trees is equivalent to message passing along tree branches!



$$m_{ji}(x_i) = \sum_{x_j}\left(\psi(x_j)\psi(x_i,x_j)\prod_{k\in N(j)\setminus i} m_{kj}(x_j)\right)$$

The message passing protocol:

- A two-pass algorithm:



Belief Propagation (SP-algorithm): Sequential implementation

SUM-PRODUCT($\mathcal{T}$, E)
    EVIDENCE(E)
    f = CHOOSEROOT($\mathcal{V}$)
    for $e \in \mathcal{N}(f)$
        COLLECT(f, e)
    for $e \in \mathcal{N}(f)$
        DISTRIBUTE(f, e)
    for $i \in \mathcal{V}$
        COMPUTEMARGINAL(i)

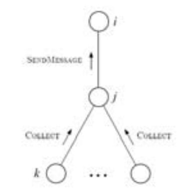
EVIDENCE(E)
    for $i \in E$
        $\psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$
    for $i \notin E$
        $\psi^E(x_i) = \psi(x_i)$

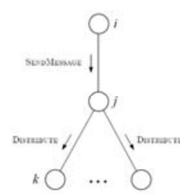
COLLECT(i, j)
    for $k \in \mathcal{N}(j) \backslash i$
        COLLECT(j, k)
    SENDMESSAGE(j, i)

DISTRIBUTE(i, j)
    SENDMESSAGE(i, j)
    for $k \in \mathcal{N}(j) \backslash i$
        DISTRIBUTE(j, k)

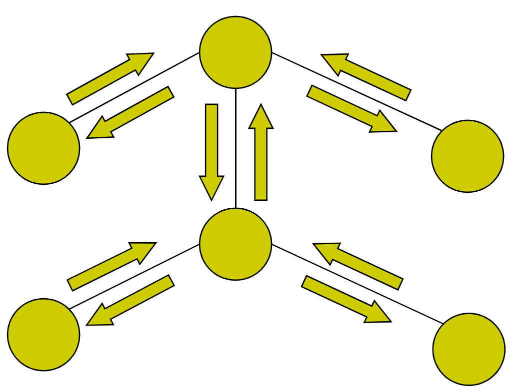
SENDMESSAGE(j, i)
$$m_{ji}(x_i) = \sum_{x_j} (\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \backslash i} m_{kj}(x_j))$$

COMPUTEMARGINAL(i)
$$p(x_i) \propto \psi^E(x_i) \prod_{j \in \mathcal{N}(i)} m_{ji}(x_i)$$





Belief Propagation (SP-algorithm): Parallel synchronous implementation



- For a node of degree d, whenever messages have arrived on any subset of d-1 node, compute the message for the remaining edge and send!
  - A pair of messages have been computed for each edge, one for each direction
  - All incoming messages are eventually computed for each node

Correctness of BP on tree

- Collollary: the synchronous implementation is "non-blocking"
- Thm: The Message Passage Guarantees obtaining all marginals in the tree

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

- What about non-tree?

Inference on general GM

- Now, what if the GM is not a tree-like graph?
- Can we still directly run message
  message-passing protocol along its edges?
- For non-trees, we do not have the guarantee that message-passing will be consistent!
- Then what?
  - Construct a graph data-structure from P that has a tree structure, and run message-passing on it!

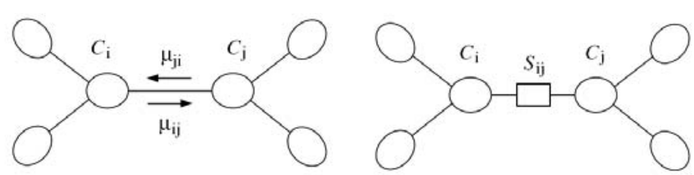
→ Junction tree algorithm

A Sketch of the Junction Tree Algorithm

- **The algorithm**
  - Construction of junction trees --- a special **clique tree**
  - Propagation of probabilities --- a message-passing protocol
- Results in marginal probabilities of all cliques --- solves all queries in a single run
- A **generic** exact inference algorithm for any GM
- **Complexity**: exponential in the size of the maximal clique --- a good elimination order often leads to small maximal clique, and hence a good (i.e., thin) JT
- Many well-known algorithms are special cases of JT
  - Forward-backward, Kalman filter, Peeling, Sum-Product ...

The Shafer Shenoy Algorithm

- Shafer-Shenoy algorithm



(a) (b)

  - Message from clique i to clique j :

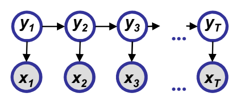
$$\mu_{i\to j} = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq j} \mu_{k\to i}(S_{ki})$$

  - Clique marginal

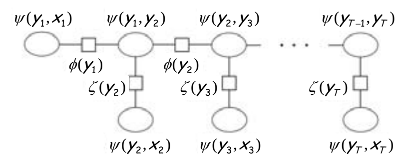
$$p(C_i) \propto \psi_{C_i} \prod_{k} \mu_{k\to i}(S_{ki})$$

The Junction tree algorithm for HMM

- A junction tree for the HMM

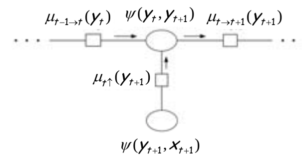


$\Rightarrow$



- Rightward pass

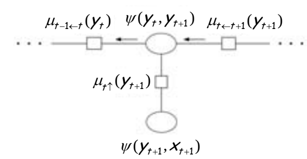
$$\mu_{t\to t+1}(y_{t+1}) = \sum_{y_t} \psi(y_t, y_{t+1})\mu_{t-1\to t}(y_t)\mu_{t\uparrow}(y_{t+1})$$
$$= \sum_{y_t} p(y_{t+1} \mid y_t)\mu_{t-1\to t}(y_t)\, p(x_{t+1} \mid y_{t+1})$$
$$= p(x_{t+1} \mid y_{t+1})\sum_{y_t} a_{y_t, y_{t+1}}\mu_{t-1\to t}(y_t)$$



  - This is exactly the *forward algorithm*!

- Leftward pass …

$$\mu_{t-1\leftarrow t}(y_t) = \sum_{y_{t+1}} \psi(y_t, y_{t+1})\mu_{t\leftarrow t+1}(y_{t+1})\mu_{t\uparrow}(y_{t+1})$$
$$= \sum_{y_{t+1}} p(y_{t+1} \mid y_t)\mu_{t\leftarrow t+1}(y_{t+1})\, p(x_{t+1} \mid y_{t+1})$$



  - This is exactly the *backward algorithm*!

Summary

- The simple Eliminate algorithm captures the key algorithmic Operation underlying probabilistic inference:
  --- That of taking a sum over product of potential functions

- The computational complexity of the Eliminate algorithm can be reduced to purely graph-theoretic considerations.

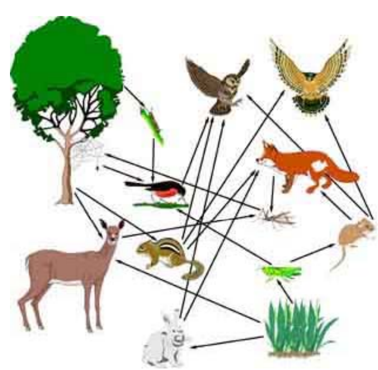
- This graph interpretation will also provide hints about how to design improved inference algorithms

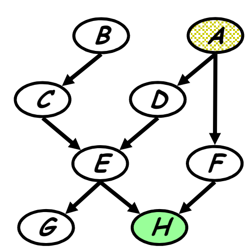
- What can we say about the overall computational complexity of the algorithm? In particular, how can we control the "size" of the summands that appear in the sequence of summation operation.

Extra reading:

From Elimination to JT on a general Bayesian network

A food web

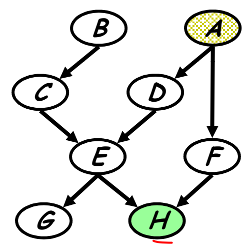




What is the probability that hawks are leaving given that the grass condition is poor?

Example: Variable Elimination

- Query: *P(A |h)*
  - Need to eliminate: *B,C,D,E,F,G,H*



- Initial factors:

$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$

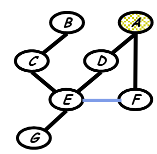
- Choose an elimination order: *H,G,F,E,D,C,B*

- Step 1:
  - **Conditioning** (fix the evidence node (i.e., *h*) on its observed value (i.e., $\tilde{h}$)):

$$m_h(e,f) = p(h=\tilde{h}\,|\,e,f)$$

  - This step is isomorphic to a marginalization step:

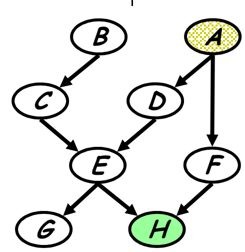
$$m_h(e,f) = \sum_h p(h|e,f)\delta(h=\tilde{h})$$



63

Example: Variable Elimination

- Query: *P(B |h)*
  - Need to eliminate: *B,C,D,E,F,G*



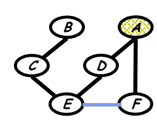
- Initial factors:

$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$

- Step 2: Eliminate *G*
  - compute

$$m_g(e) = \sum_g p(g|e) = 1$$

$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_g(e)m_h(e,f)$$
$$= P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$

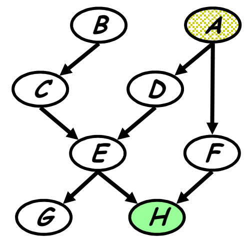


64

Example: Variable Elimination

- Query: *P(B | h)*
  - Need to eliminate: *B,C,D,E,F*
- Initial factors:

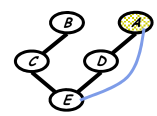
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$



- Step 3: Eliminate *F*
  - compute

$$m_f(e,a) = \sum_f p(f|a)m_h(e,f)$$

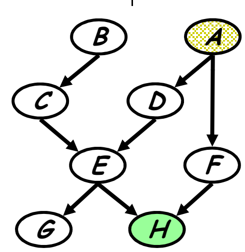
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)m_f(a,e)$$



Example: Variable Elimination

- Query: *P(B | h)*
  - Need to eliminate: *B,C,D,E*
- Initial factors:

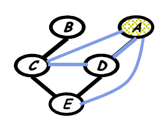
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)m_f(a,e)$$



- Step 4: Eliminate *E*
  - compute

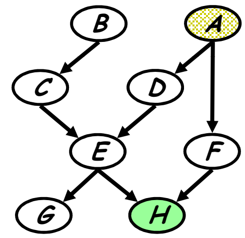
$$m_e(a,c,d) = \sum_e p(e|c,d)m_f(a,e)$$

$$\Rightarrow P(a)P(b)P(c|b)P(d|a)m_e(a,c,d)$$



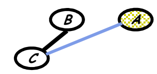
Example: Variable Elimination

- Query: *P(B | h)*
  - Need to eliminate: *B,C,D*
- Initial factors:



$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)m_e(a,c,d)$$

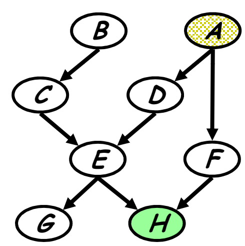
- Step 5: Eliminate *D*
  - compute $$m_d(a,c) = \sum_d p(d|a)m_e(a,c,d)$$



$$\Rightarrow P(a)P(b)P(c|d)m_d(a,c)$$

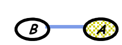
Example: Variable Elimination

- Query: *P(B | h)*
  - Need to eliminate: *B,C*
- Initial factors:



$$P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)m_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c|d)m_d(a,c)$$

- Step 6: Eliminate *C*
  - compute $$m_c(a,b) = \sum_c p(c|b)m_d(a,c)$$

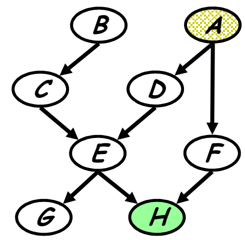


$$\Rightarrow P(a)P(b)P(c|d)m_d(a,c)$$

Example: Variable Elimination

- Query: *P(B | h)*
  - Need to elimінate: *B*
- Initial factors:

$$P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)m_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c|d)m_d(a,c)$$
$$\Rightarrow P(a)P(b)m_c(a,b)$$



- Step 7: Eliminate *B*
  - compute $$m_b(a) = \sum_b p(b)m_c(a,b)$$

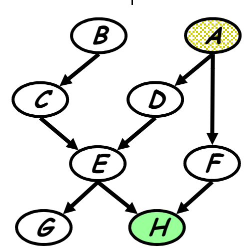
$$\Rightarrow P(a)m_b(a)$$

© Eric Xing @ CMU, 2006-2011 69

Example: Variable Elimination

- Query: *P(B | h)*
  - Need to eliminate: *B*
- Initial factors:

$$P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)m_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c|d)m_d(a,c)$$
$$\Rightarrow P(a)P(b)m_c(a,b)$$
$$\Rightarrow P(a)m_b(a)$$



- Step 8: Wrap-up $$p(a,\tilde{h}) = p(a)m_b(a), \quad p(\tilde{h}) = \sum_a p(a)m_b(a)$$

$$\Rightarrow P(a|\tilde{h}) = \frac{p(a)m_b(a)}{\sum_a p(a)m_b(a)}$$

© Eric Xing @ CMU, 2006-2011 70

Complexity of variable elimination

- Suppose in one elimination step we compute

$$m_x(y_1,\ldots,y_k) = \sum_x m'_x(x, y_1,\ldots,y_k)$$

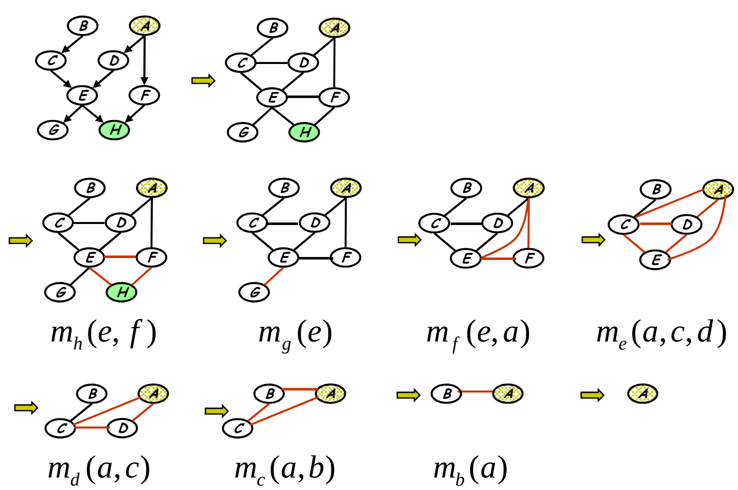
$$m'_x(x, y_1,\ldots,y_k) = \prod_{i=1}^{k} m_i(x, \mathbf{y}_{c_i})$$

This requires

- $k \bullet |\mathrm{Val}(X)| \bullet \prod_i |\mathrm{Val}(\mathbf{Y}_{C_i})|$ multiplications
  - For each value of $x, y_1, \ldots, y_k$, we do k multiplications
- $|\mathrm{Val}(X)| \bullet \prod_i |\mathrm{Val}(\mathbf{Y}_{C_i})|$ additions
  - For each value of $y_1, \ldots, y_k$, we do $|Val(X)|$ additions

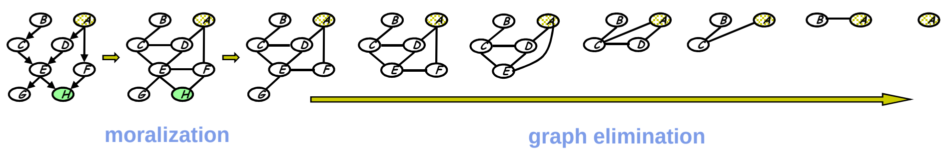
Complexity is **exponential** in number of variables in the intermediate factor

Elimination Cliques



Understanding Variable Elimination

- A graph elimination algorithm

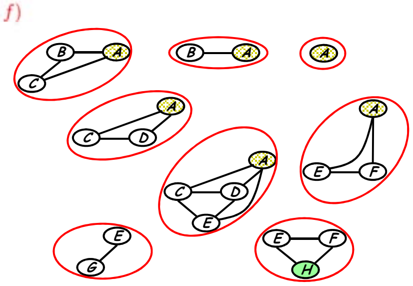


- Intermediate terms correspond to the cliques resulted from elimination
  - “good” elimination orderings lead to **small cliques** and hence reduce complexity
    (what will happen if we eliminate "e" first in the above graph?)
  - finding the optimum ordering is NP-hard, but for many graph optimum or near-optimum can often be heuristically found
- Applies to undirected GMs

Elimination Clique

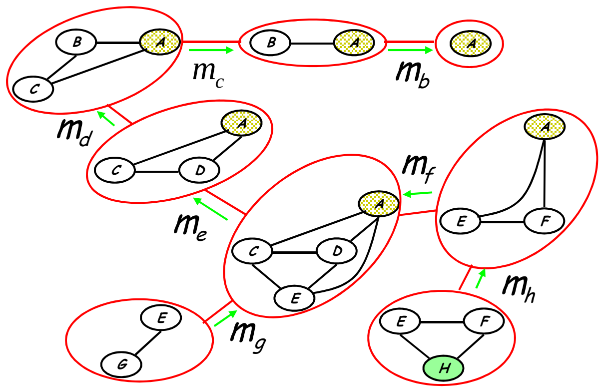
- Recall that Induced dependency during marginalization is captured in elimination cliques
  - Summation <-> elimination
  - Intermediate term <-> elimination clique

$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)\phi_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)\phi_g(e)\phi_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)\phi_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)\phi_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c|b)\phi_d(a,c)$$
$$\Rightarrow P(a)P(b)\phi_c(a,b)$$
$$\Rightarrow P(a)\phi_b(a)$$
$$\Rightarrow \phi(a)$$



  - Can this lead to an generic inference algorithm?

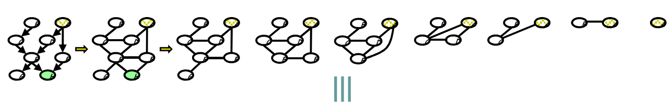
A Clique Tree

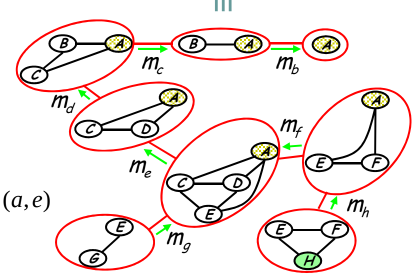


$$m_e(a,c,d) = \sum_e p(e \mid c,d) m_g(e) m_f(a,e)$$

From Elimination to Message Passing

- Elimination ≡ message passing on a **clique tree**



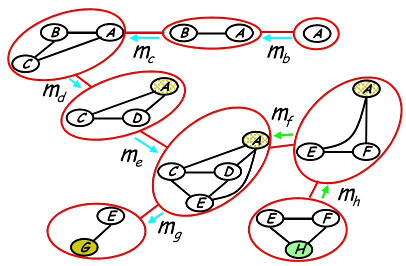


$$m_e(a,c,d) = \sum_e p(e \mid c,d) m_g(e) m_f(a,e)$$

- Messages can be reused

From Elimination to Message Passing

- Elimination ≡ message passing on a **clique tree**
  - **Another query ...**

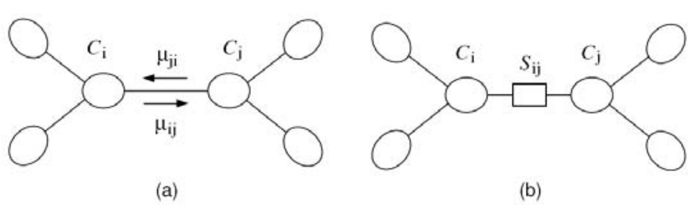


- Messages m_f and m_h are reused, others need to be recomputed

 77

The Shafer Shenoy Algorithm

- Shafer-Shenoy algorithm



- Message from clique i to clique j :

$$\mu_{i\to j} = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq j} \mu_{k\to i}(S_{ki})$$

- Clique marginal

$$p(C_i) \propto \psi_{C_i} \prod_{k} \mu_{k\to i}(S_{ki})$$

 78

A Sketch of the Junction Tree Algorithm

- **The algorithm**
  - Construction of junction trees --- a special **clique tree**
  - Propagation of probabilities --- a message-passing protocol
- Results in marginal probabilities of all cliques --- solves all queries in a single run
- A **generic** exact inference algorithm for any GM
- **Complexity**: exponential in the size of the maximal clique --- a good elimination order often leads to small maximal clique, and hence a good (i.e., thin) JT
- Many well-known algorithms are special cases of JT
  - Forward-backward, Kalman filter, Peeling, Sum-Product ...