Graphical Models

Yang Feng

An introduction to Graphical Models – Michael Jordan
Scalable Machine Learning : Graphical Models
--Alex Amola
PRML – Christopher Bishop

Graphical Models

- Graphical Models are a marriage between graph theory and probability theory
- There are two kinds of graphical models: directed graphical models (Bayesian networks) and undirected graphical models (Markov Random fields)
- Alternative names for graphical models: belief networks, Bayesian networks, probabilistic independent networks, Markov random fields, loglinear models, influence diagrams

Graphical Models

- A key insight from the graphical model point of view:

  It is not necessary to learn that which can be inferred

- The weights in a network make local assertions about the relationships between neighboring nodes
- Inference algorithms turn these local assertions into global assertions about the relationships between nodes
  - e.g., correlations between hidden units conditional on an input-output pair
  - e.g., the probability of an input vector given an output vector

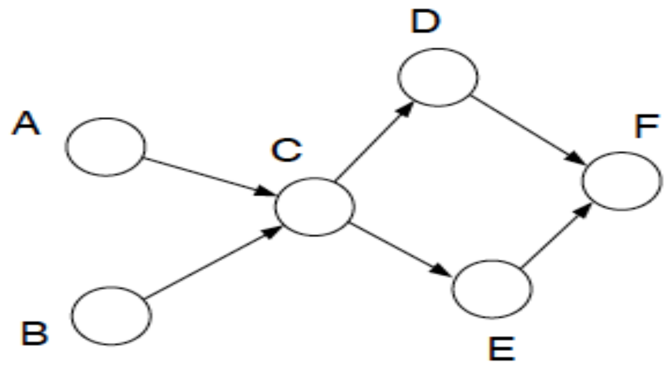
- This is achieved by associating a joint probability distribution with the network

Outline

- Directed graphical models
  - Basics
  - Sum-production Algorithm (Exact Inference)
- Undirected graphical models
  - Basics
  - Junction Tree Algorithm (Exact Inference)
- Learning
- Inference

Rresentation

- Consider an arbitrary directed (acyclic) graph, where each node in the graph corresponds to a random variable (scalar or vector):



- There is no a priori need to designate units as "inputs," "outputs" or "hidden"

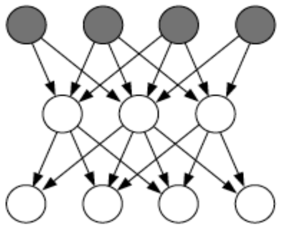
- We want to associate a probability distribution $P(A, B, C, D, E, F)$ with this graph, and we want all of our calculations to respect this distribution

e.g.,

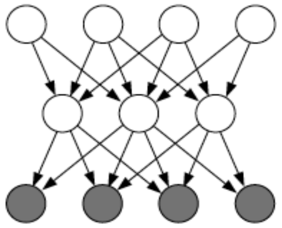
$$P(F|A, B) = \frac{\Sigma_C \Sigma_D \Sigma_E P(A, B, C, D, E, F)}{\Sigma_C \Sigma_D \Sigma_E \Sigma_F P(A, B, C, D, E, F)}$$

Some ways to use Directed GM

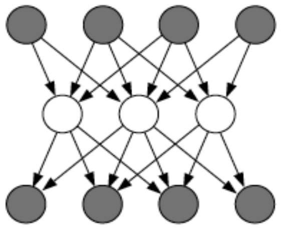
Prediction:



Diagnosis, control, optimization:



Supervised learning:



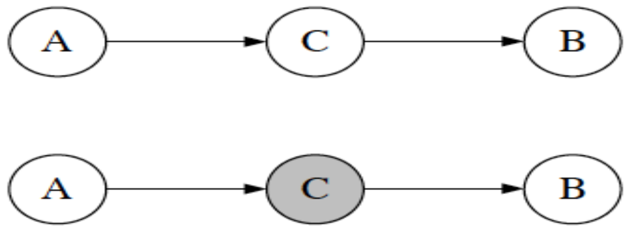
- we want to marginalize over the unshaded nodes in each case (i.e., integrate them out from the joint probability)
- "unsupervised learning" is the general case

Specification of a GM

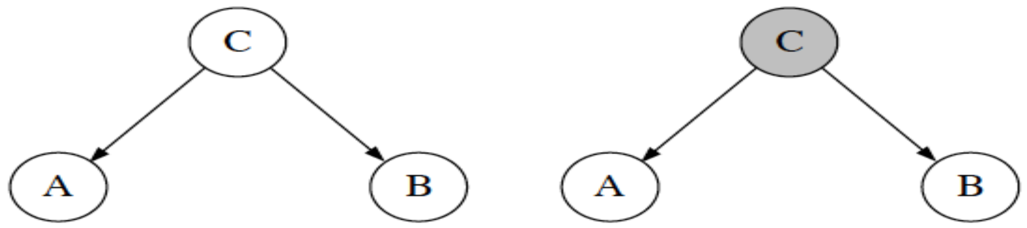
- There are two components to any graphical model:
  - the *qualitative* specification
  - the *quantitative* specification

- 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 like a certain architecture (e.g., a layered graph)

Qualitative Specification

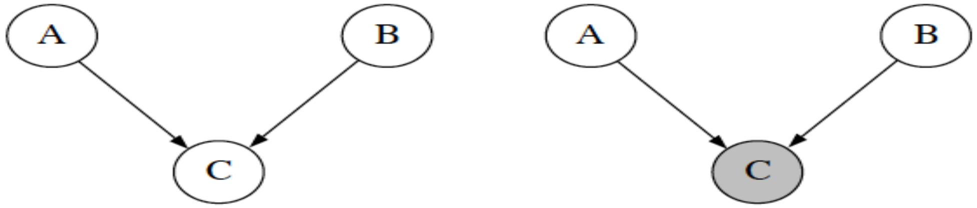


- A and B are marginally *dependent*
- A and B are conditionally *independent*



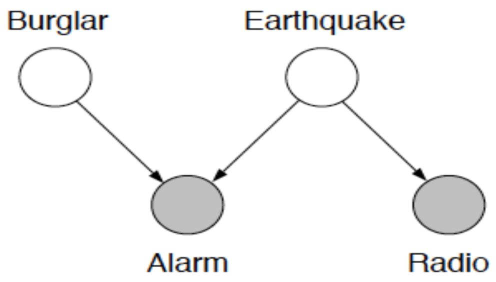
- A and B are marginally *dependent*
- A and B are conditionally *independent*

Qualitative Specification (cont)



- A and B are marginally *independent*
- A and B are conditionally *dependent*

- This is the interesting case...



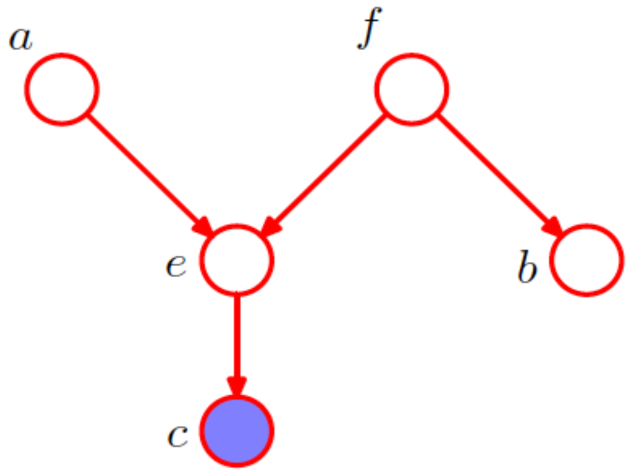
- All connections (in both directions) are "excitatory"
- But an increase in "activation" of Earthquake leads to a decrease in "activation" of Burglar
- Where does the "inhibition" come from?

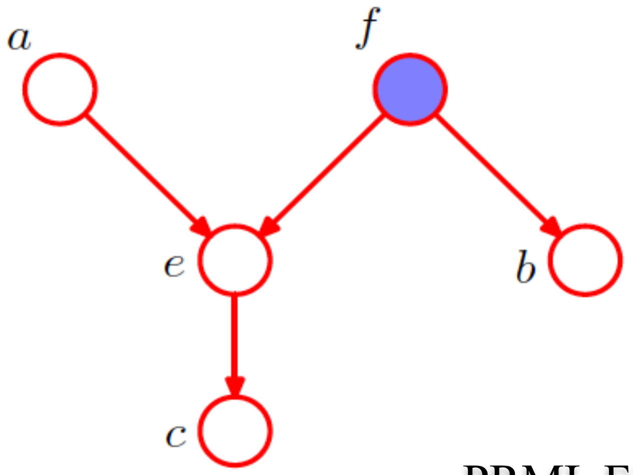
D-separation (Bayes Ball)

- A is d-separated from B by C if all possible paths from any node in A to any node in B are blocked by C.
- A path is blocked if it includes such that either
  - the arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set C, or
  - the arrows meet head-to-head at the node, and neither the node, nor any of its descendants, is in the set C.

$A \perp\!\!\!\perp B \mid C$ A is d-separated from B by C

D-separation for Directed GM (cont)





PRML Figure 8.22

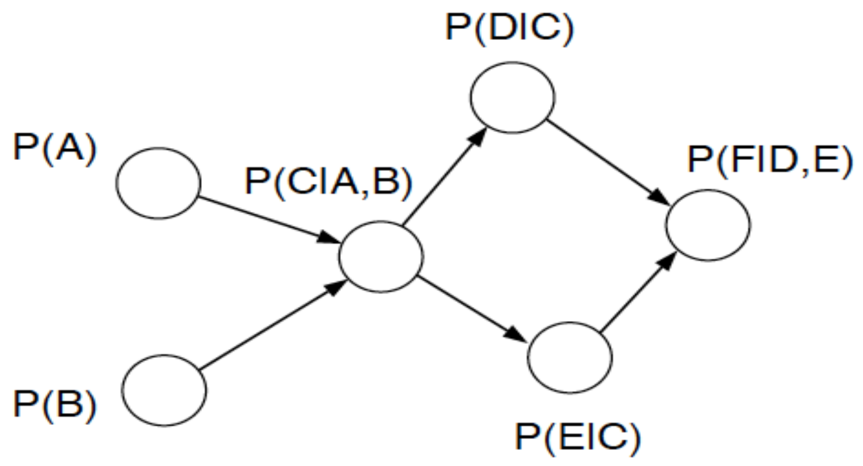
$a \perp\!\!\!\perp b \mid c$ ✗

$a \perp\!\!\!\perp b \mid f$ ✓

Bayes ball

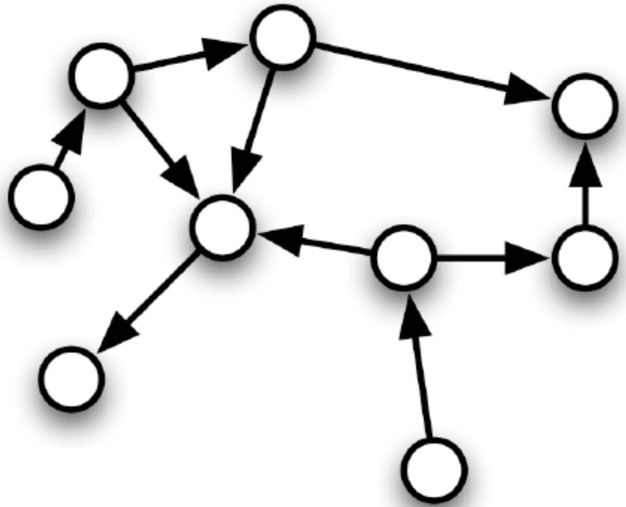
Quantitative Specification

- *Question*: how do we specify a joint distribution over the nodes in the graph?
- *Answer*: associate a conditional probability with each node:



and take the product of the *local* probabilities to yield the *global* probabilities

Quantitative Specification (cont)



- Joint probability distribution

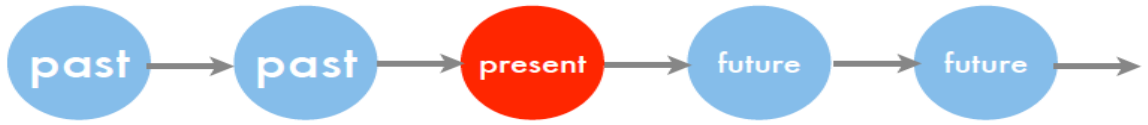
$$p(x) = \prod_i p(x_i | x_{\text{parents(i)}})$$

- Parameter estimation
  - If x is fully observed the likelihood breaks up

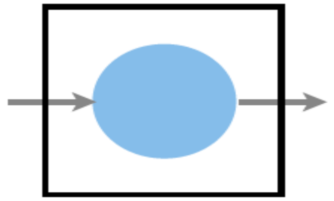
$$\log p(x|\theta) = \sum_i \log p(x_i | x_{\text{parents(i)}}, \theta)$$

  - If x is partially observed things get interesting maximization, EM, variational, sampling ...

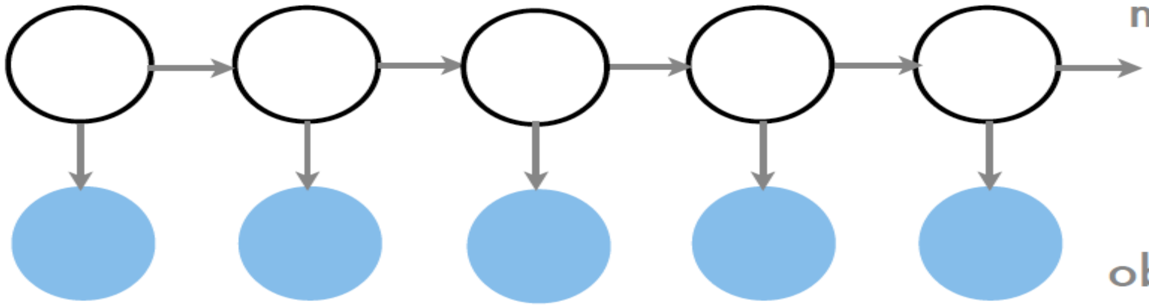
Markov Chain

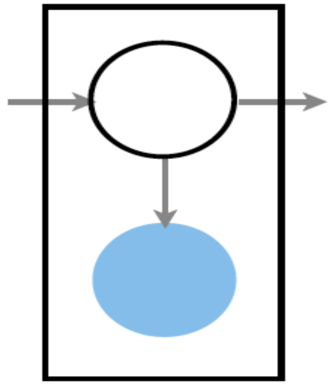


Plate



Hidden Markov Chain



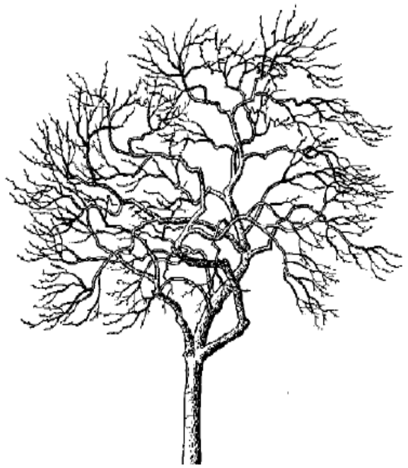


user model for traversal through search results

Outline

- Directed graphical models
  - Basics
  - Sum-production Algorithm (Exact Inference)
- Undirected graphical models
  - Basics
  - Junction Tree Algorithm (Exact Inference)
- Learning
- Inference

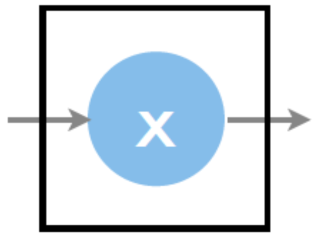
Chains and Trees



Chains

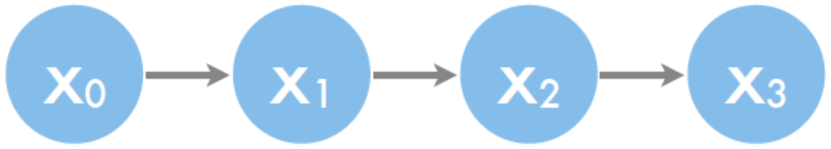
$$p(x;\theta) = p(x_0;\theta)\prod_{i=1}^{n-1} p(x_{i+1}|x_i;\theta)$$

$$\begin{tikzcd} x_0 \arrow[r] & x_1 \arrow[r] & x_2 \arrow[r] & x_3 \end{tikzcd}$$

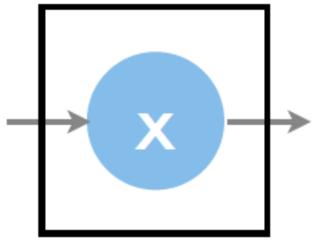


Chains (cont)

$$p(x;\theta) = p(x_0;\theta)\prod_{i=1}^{n-1} p(x_{i+1}|x_i;\theta)$$

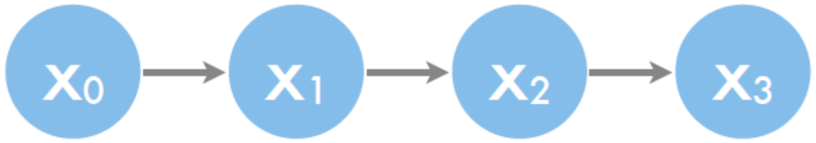


$$p(x_i) = \sum_{x_0,\dots x_{i-1},x_{i+1}\dots x_n} \underbrace{p(x_0)}_{:=l_0(x_0)} \prod_{j=1}^{n} p(x_j|x_{j-1})$$



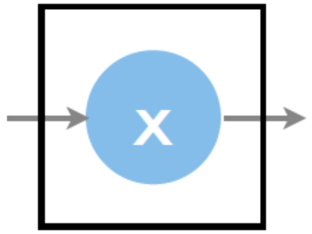
Chains (cont)

$$p(x;\theta) = p(x_0;\theta)\prod_{i=1}^{n-1} p(x_{i+1}|x_i;\theta)$$



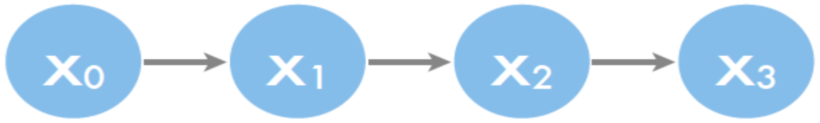
$$p(x_i) = \sum_{x_0,\ldots x_{i-1},x_{i+1}\ldots x_n} \underbrace{p(x_0)}_{:=l_0(x_0)} \prod_{j=1}^{n} p(x_j|x_{j-1})$$

$$= \sum_{x_1,\ldots x_{i-1},x_{i+1}\ldots x_n} \underbrace{\sum_{x_0}[l_0(x_0)p(x_1|x_0)]}_{:=l_1(x_1)} \prod_{j=2}^{n} p(x_j|x_{j-1})$$



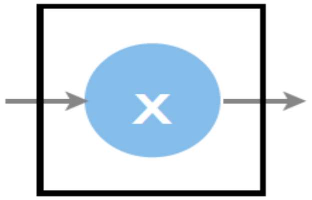
Chains (cont)

$$p(x;\theta) = p(x_0;\theta)\prod_{i=1}^{n-1} p(x_{i+1}|x_i;\theta)$$



$$p(x_i) = \sum_{x_0,\ldots x_{i-1},x_{i+1}\ldots x_n} \underbrace{p(x_0)}_{:=l_0(x_0)} \prod_{j=1}^{n} p(x_j|x_{j-1})$$

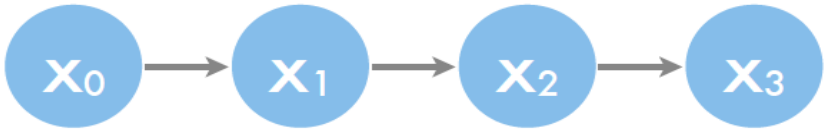
$$= \sum_{x_1,\ldots x_{i-1},x_{i+1}\ldots x_n} \underbrace{\sum_{x_0}[l_0(x_0)p(x_1|x_0)]}_{:=l_1(x_1)} \prod_{j=2}^{n} p(x_j|x_{j-1})$$



$$= \sum_{x_2,\ldots x_{i-1},x_{i+1}\ldots x_n} \underbrace{\sum_{x_1}[l_1(x_1)p(x_2|x_1)]}_{:=l_2(x_2)} \prod_{j=3}^{n} p(x_j|x_{j-1})$$

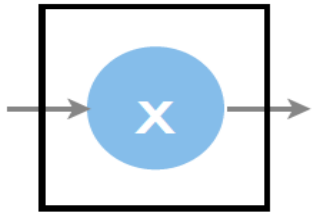
Chains (cont)

$$p(x;\theta) = p(x_0;\theta)\prod_{i=1}^{n-1} p(x_{i+1}|x_i;\theta)$$



$$p(x_i) = l_i(x_i)\sum_{x_{i+1}\ldots x_n}\prod_{j=i}^{n-1} p(x_{j+1}|x_j)$$

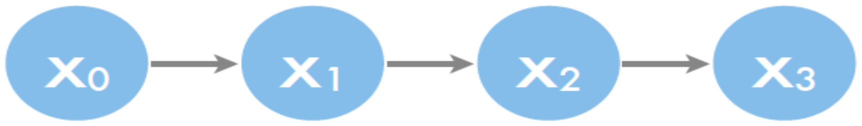
$$= l_i(x_i)\sum_{x_{i+1}\ldots x_{n-1}}\prod_{j=i}^{n-2} p(x_{j+1}|x_j)\underbrace{\sum_{x_n} p(x_n|x_{n-1})}_{:=r_{n-1}(x_{n-1})}$$



$$= l_i(x_i)\sum_{x_{i+1}\ldots x_{n-2}}\prod_{j=i}^{n-3} p(x_{j+1}|x_j)\underbrace{\sum_{x_{n-1}} p(x_{n-1}|x_{n-2})r_{n-1}(x_{n-1})}_{:=r_{n-2}(x_{n-2})}$$

Chains (cont)

$$p(x;\theta) = p(x_0;\theta)\prod_{i=1}^{n-1} p(x_{i+1}|x_i;\theta)$$



- **Forward recursion**

$$l_0(x_0) := p(x_0) \text{ and } l_i(x_i) := \sum_{x_{i-1}} l_{i-1}(x_{i-1})p(x_i|x_{i-1})$$

- **Backward recursion**

$$r_n(x_n) := 1 \text{ and } r_i(x_i) := \sum_{x_{i+1}} r_{i+1}(x_{i+1})p(x_{i+1}|x_i)$$

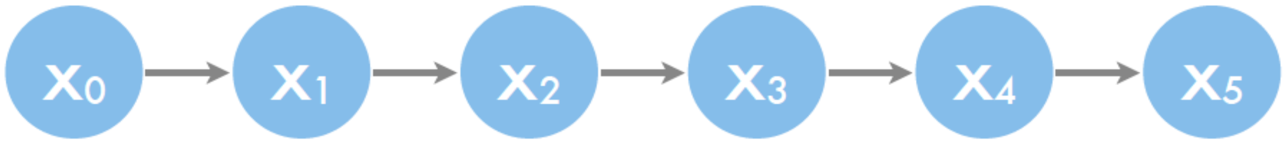
- **Marginalization & conditioning**

$$p(x_i) = l_i(x_i)r_i(x_i)$$

$$p(x_{-i}|x_i) = \frac{p(x)}{p(x_i)}$$

$$p(x_i, x_{i+1}) = l_i(x_i)p(x_{i+1}|x_i)r_i(x_{i+1})$$

Chains (cont)



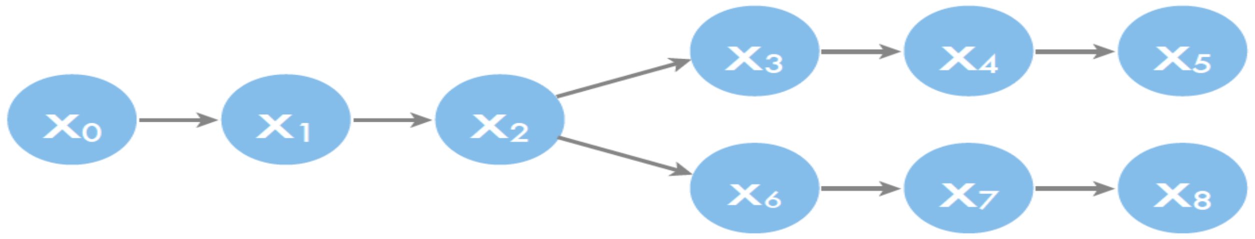
- Send forward messages starting from left node

$$m_{i-1\to i}(x_i) = \sum_{x_{i-1}} m_{i-2\to i-1}(x_{i-1}) f(x_{i-1}, x_i)$$

- Send backward messages starting from right node

$$m_{i+1\to i}(x_i) = \sum_{x_{i+1}} m_{i+2\to i+1}(x_{i+1}) f(x_i, x_{i+1})$$

Trees



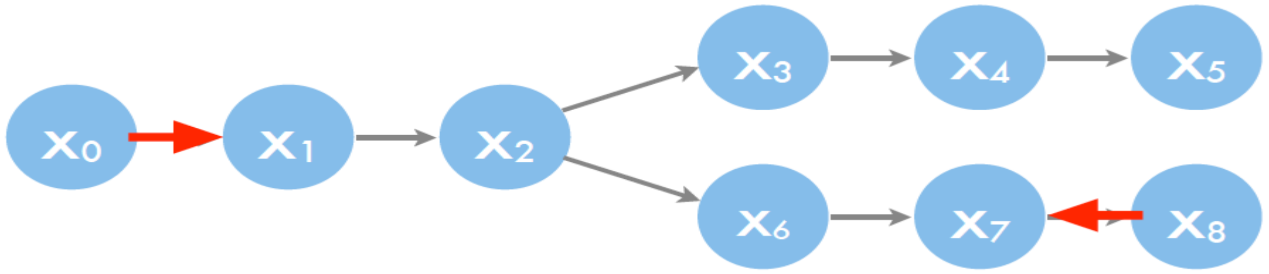
- Forward/Backward messages as normal for chain
- When we have more edges for a vertex use …

$$m_{2\to3}(x_3) = \sum_{x_2} m_{1\to2}(x_2)m_{6\to2}(x_2)f(x_2,x_3)$$

$$m_{2\to6}(x_6) = \sum_{x_2} m_{1\to2}(x_2)m_{3\to2}(x_2)f(x_2,x_6)$$

$$m_{2\to1}(x_1) = \sum_{x_2} m_{3\to2}(x_2)m_{6\to2}(x_2)f(x_1,x_2)$$

Trees (cont)



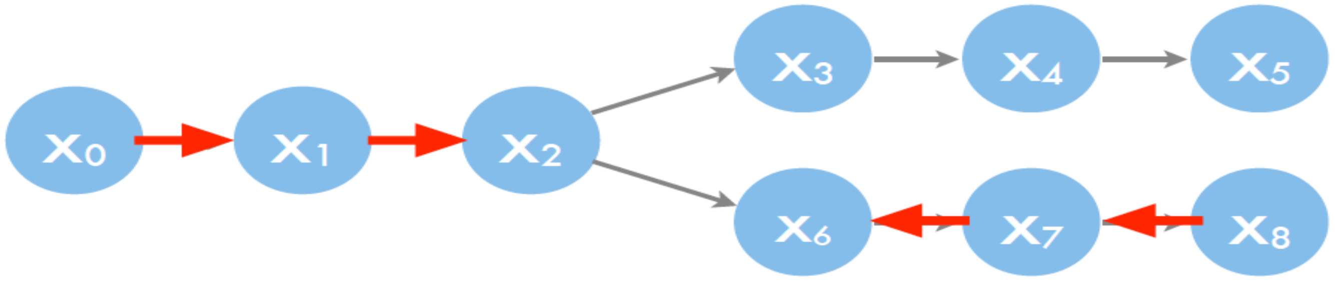
- Forward/Backward messages as normal for chain
- When we have more edges for a vertex use ...

$$m_{2\to3}(x_3) = \sum_{x_2} m_{1\to2}(x_2) m_{6\to2}(x_2) f(x_2, x_3)$$

$$m_{2\to6}(x_6) = \sum_{x_2} m_{1\to2}(x_2) m_{3\to2}(x_2) f(x_2, x_6)$$

$$m_{2\to1}(x_1) = \sum_{x_2} m_{3\to2}(x_2) m_{6\to2}(x_2) f(x_1, x_2)$$

Trees (cont)



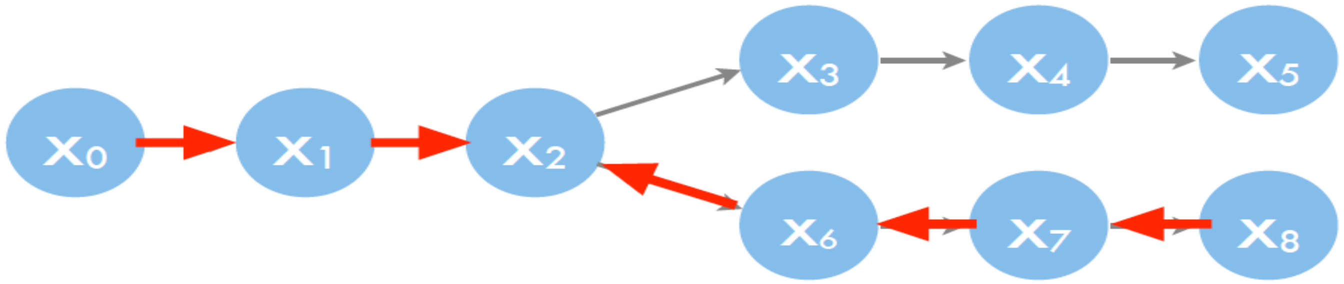
- Forward/Backward messages as normal for chain
- When we have more edges for a vertex use ...

$$m_{2\to3}(x_3) = \sum_{x_2} m_{1\to2}(x_2) m_{6\to2}(x_2) f(x_2, x_3)$$

$$m_{2\to6}(x_6) = \sum_{x_2} m_{1\to2}(x_2) m_{3\to2}(x_2) f(x_2, x_6)$$

$$m_{2\to1}(x_1) = \sum_{x_2} m_{3\to2}(x_2) m_{6\to2}(x_2) f(x_1, x_2)$$

Trees (cont)



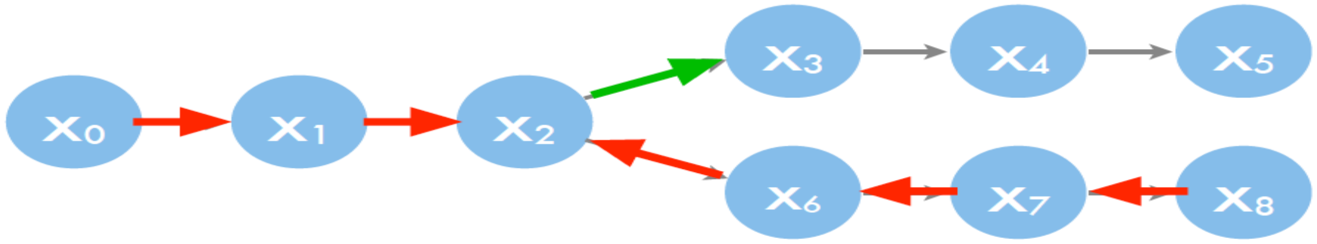
- Forward/Backward messages as normal for chain
- When we have more edges for a vertex use ...

$$m_{2\to3}(x_3) = \sum_{x_2} m_{1\to2}(x_2) m_{6\to2}(x_2) f(x_2, x_3)$$

$$m_{2\to6}(x_6) = \sum_{x_2} m_{1\to2}(x_2) m_{3\to2}(x_2) f(x_2, x_6)$$

$$m_{2\to1}(x_1) = \sum_{x_2} m_{3\to2}(x_2) m_{6\to2}(x_2) f(x_1, x_2)$$

Trees (cont)



- Forward/Backward messages as normal for chain
- When we have more edges for a vertex use ...

$$m_{2\to3}(x_3) = \sum_{x_2} m_{1\to2}(x_2) m_{6\to2}(x_2) f(x_2, x_3)$$

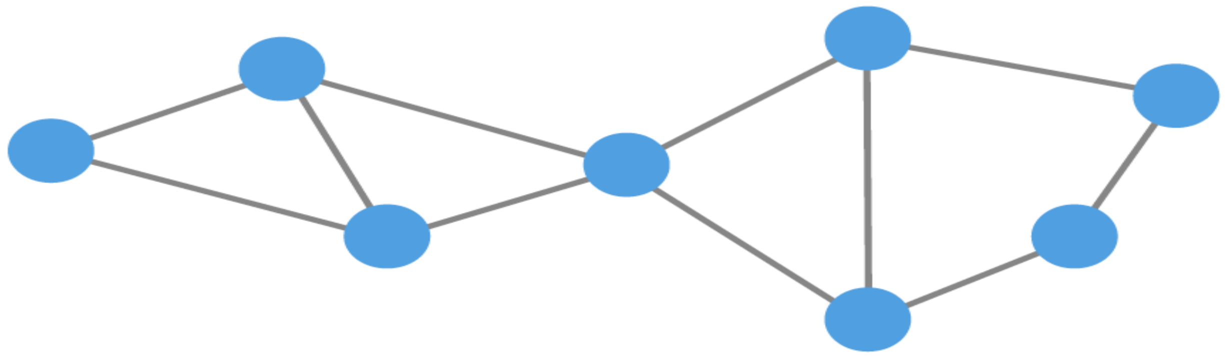
$$m_{2\to6}(x_6) = \sum_{x_2} m_{1\to2}(x_2) m_{3\to2}(x_2) f(x_2, x_6)$$

$$m_{2\to1}(x_1) = \sum_{x_2} m_{3\to2}(x_2) m_{6\to2}(x_2) f(x_1, x_2)$$

Outline

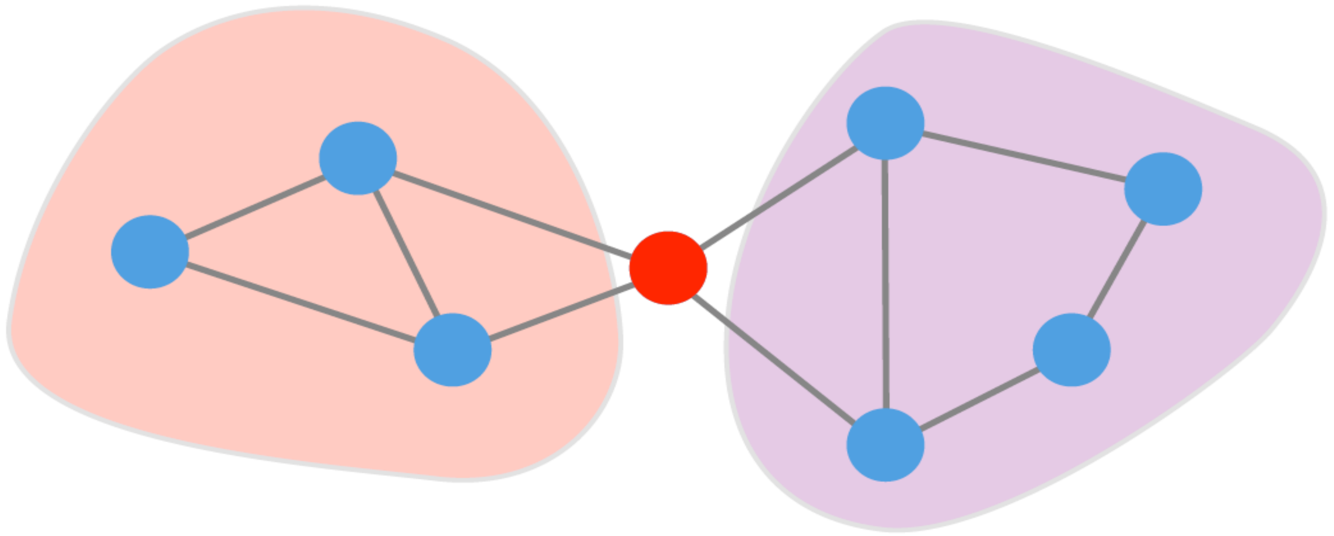
- Directed graphical models
  - Basics
  - Sum-production Algorithm (Exact Inference)
- Undirected graphical models
  - Basics
  - Junction Tree Algorithm (Exact Inference)
- Learning
- Inference

Qualitative Specification for Undirected GM



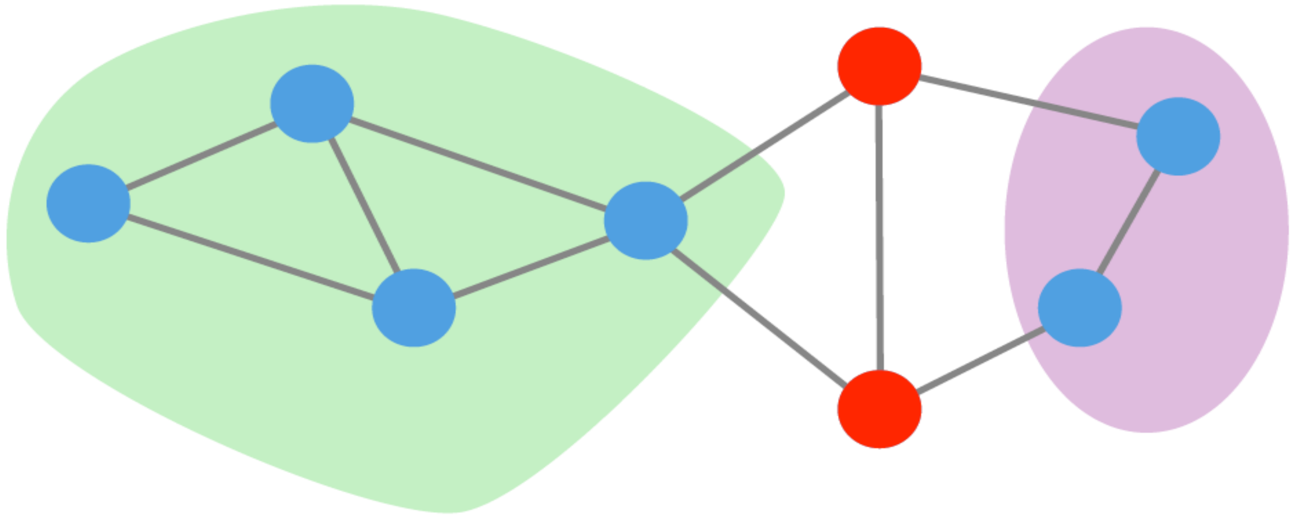
Key Concept
Observing nodes makes remainder conditionally independent

Qualitative Specification for Undirected GM



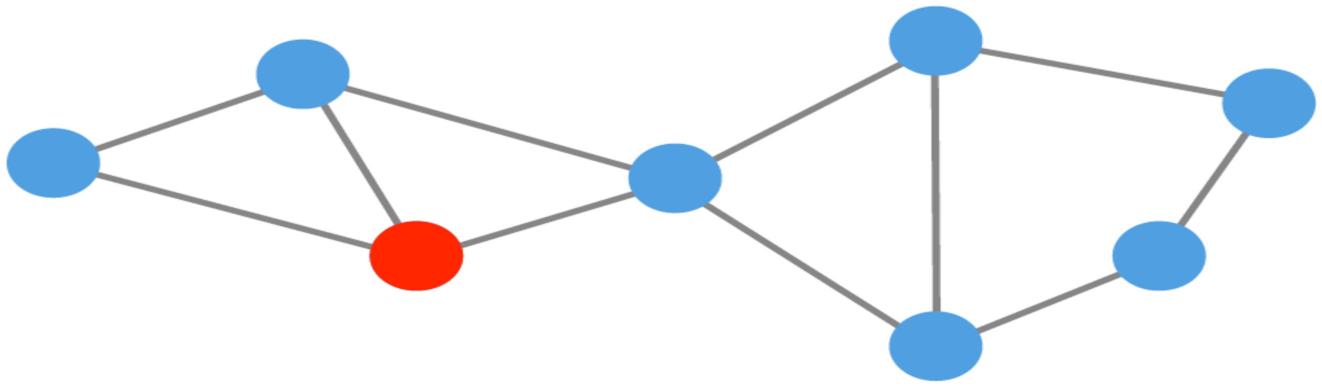
Key Concept
Observing nodes makes remainder conditionally independent

Qualitative Specification for Undirected GM



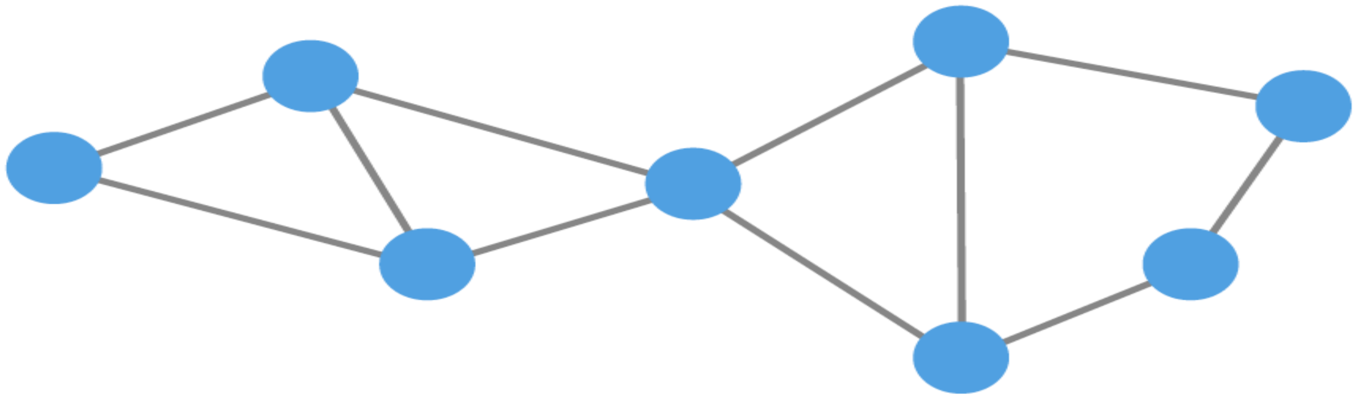
Key Concept
Observing nodes makes remainder conditionally independent

Qualitative Specification for Undirected GM



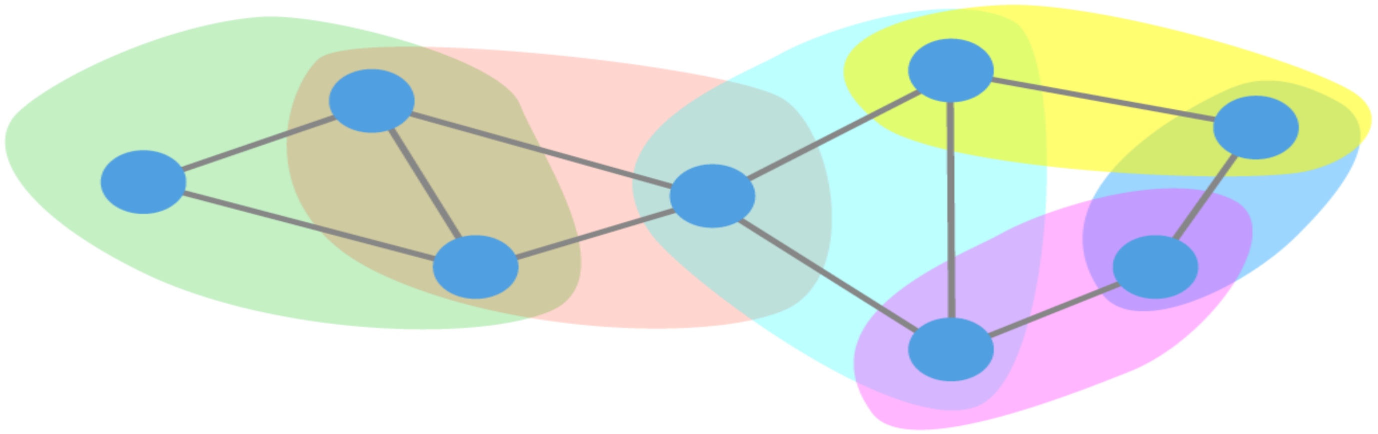
Key Concept
Observing nodes makes remainder conditionally independent

Cliques



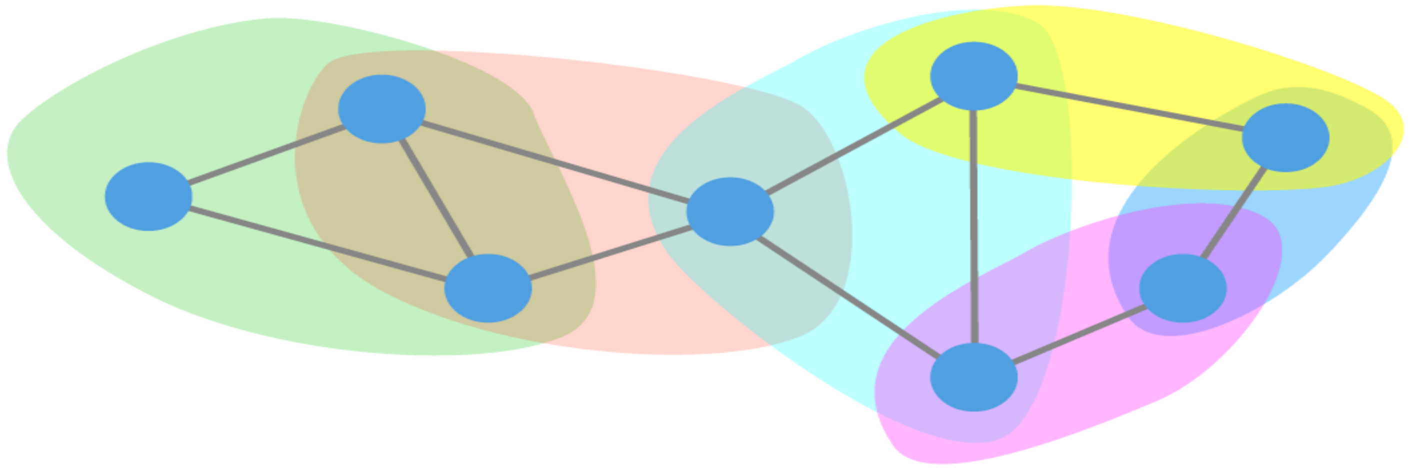
maximal fully connected subgraph

Cliques (cont)



maximal fully connected subgraph

Hammersley Clifford Theorem



If density has full support then it decomposes into products of clique potentials

$$p(x) = \prod_c \psi_c(x_c)$$

Directed vs. Undirected

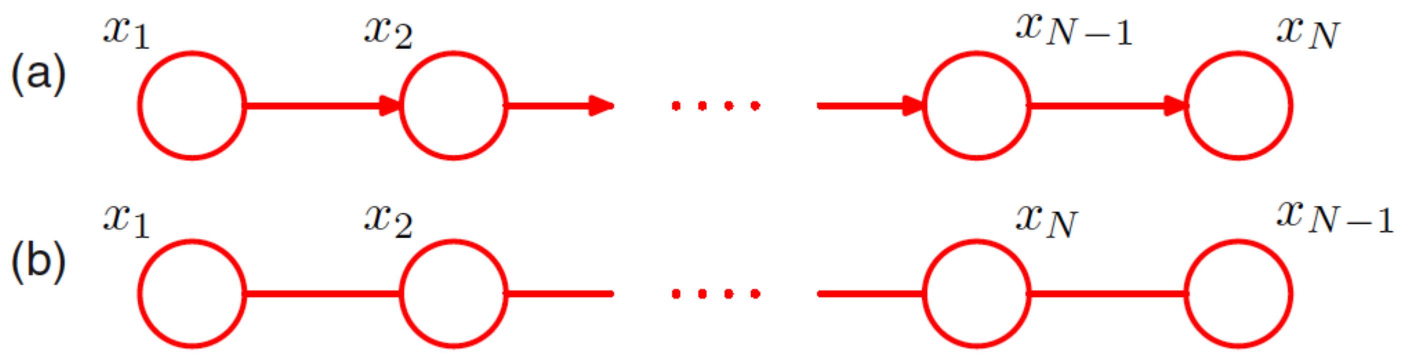
- Causal description
- Normalization automatic
- Intuitive
- Requires knowledge of dependencies
- Conditional independence tricky (Bayes Ball algorithm)

- Noncausal description (correlation only)
- Intuitive
- Easy modeling
- Normalization difficult
- Conditional independence easy to read off (graph connectivity)

Outline

- Directed graphical models
  - Basics
  - Sum-production Algorithm (Exact Inference)
- Undirected graphical models
  - Basics
  - Junction Tree Algorithm (Exact Inference)
- Learning
- Inference

Conversion from Directed to Undirected



(a) $$p(\mathbf{x}) = p(x_1)p(x_2|x_1)p(x_3|x_2)\cdots p(x_N|x_{N-1})$$

(b) $$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\cdots\psi_{N-1,N}(x_{N-1},x_N)$$

$$\begin{aligned}\psi_{1,2}(x_1,x_2) &= p(x_1)p(x_2|x_1)\\ \psi_{2,3}(x_2,x_3) &= p(x_3|x_2)\\ &\vdots\\ \psi_{N-1,N}(x_{N-1},x_N) &= p(x_N|x_{N-1})\end{aligned}$$

Moral Graph

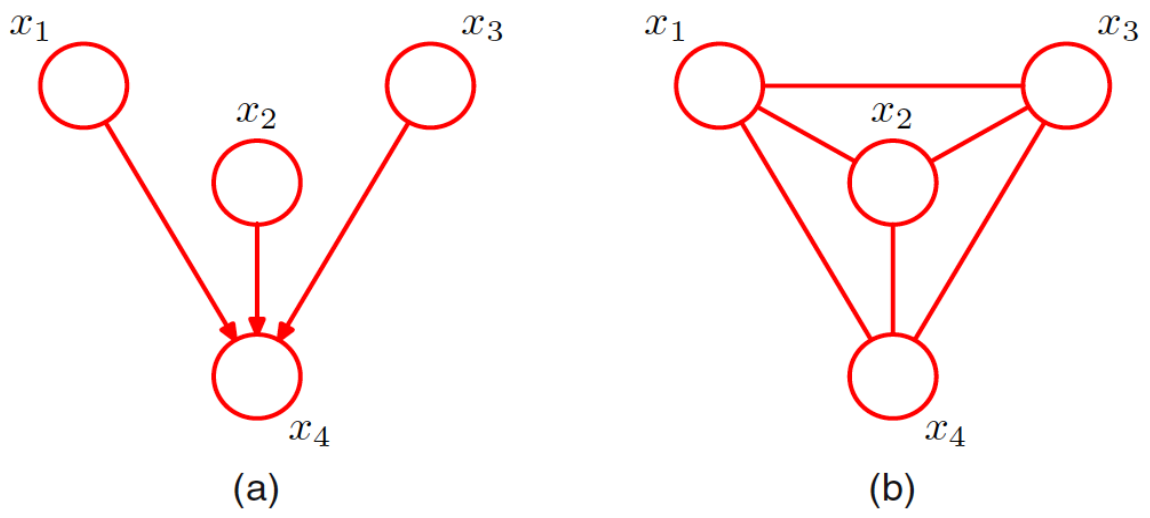
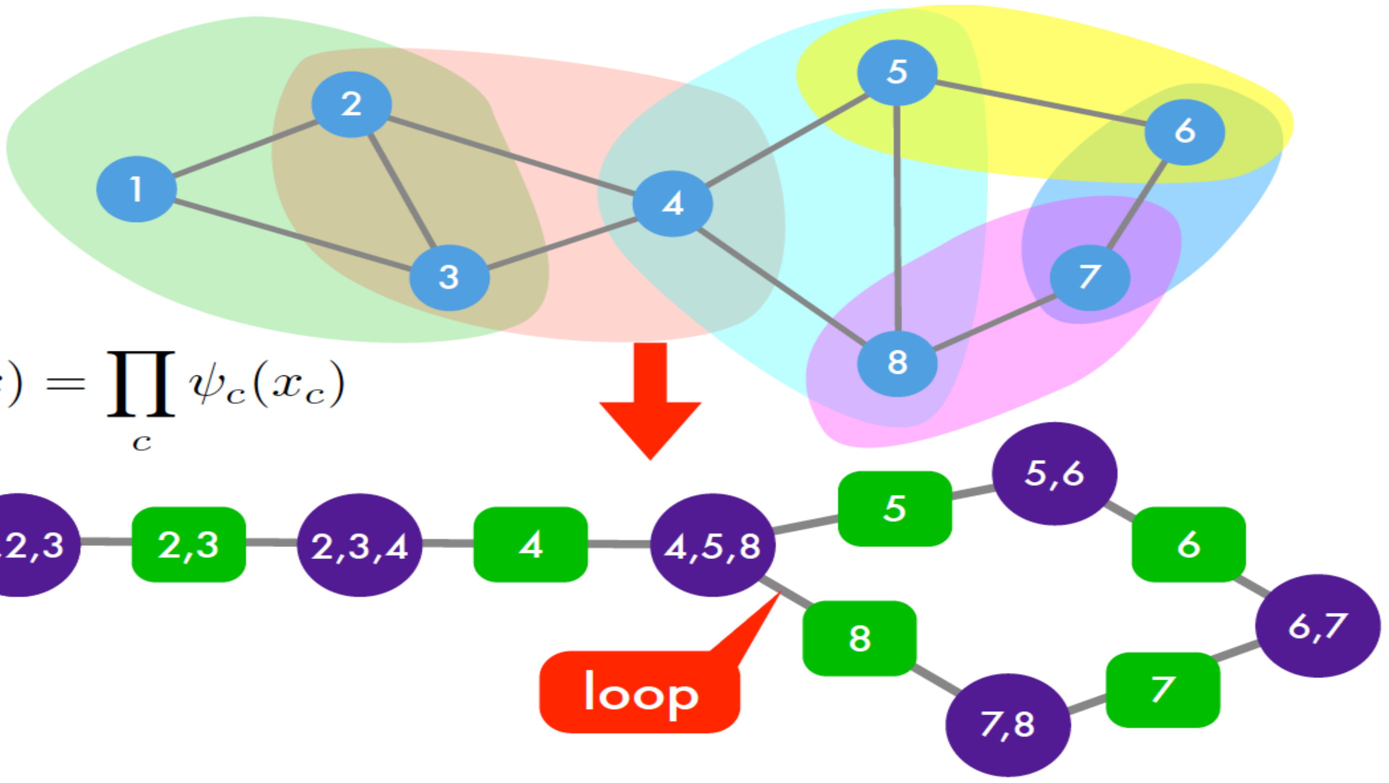


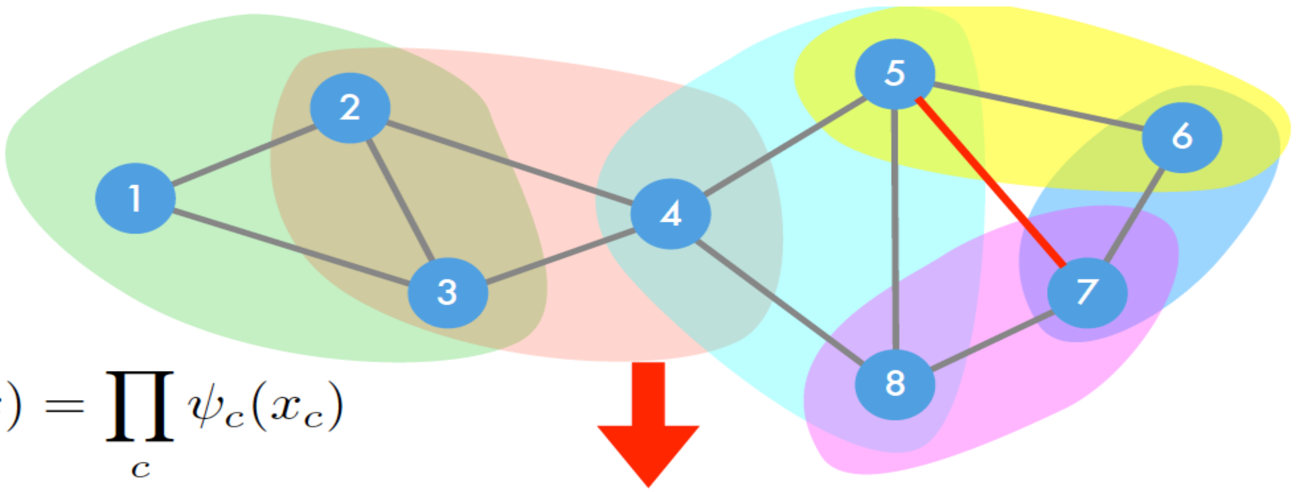
Figure 8.33 Example of a simple directed graph (a) and the corresponding moral graph (b).

Junction Tree — No Loop

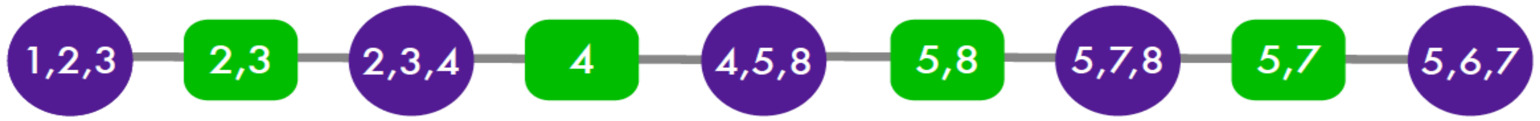


$$p(x) = \prod_c \psi_c(x_c)$$

Junction Tree -- Triangulation

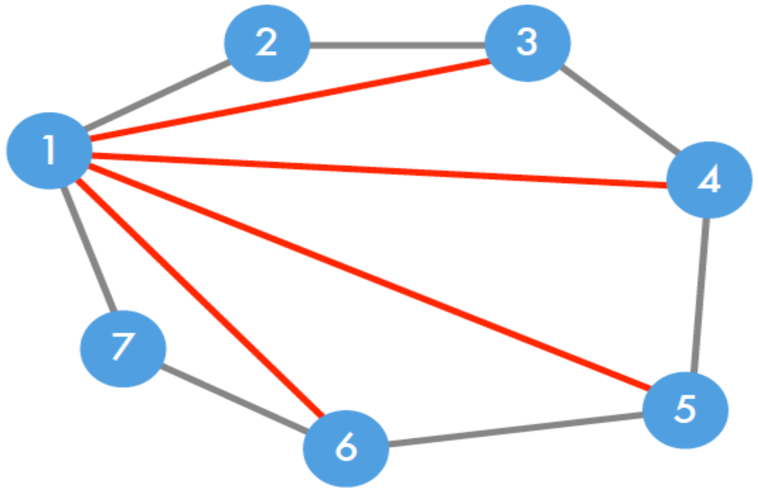


$$p(x) = \prod_c \psi_c(x_c)$$

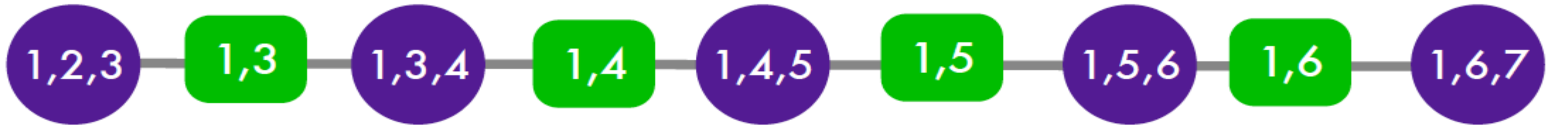


message passing possible

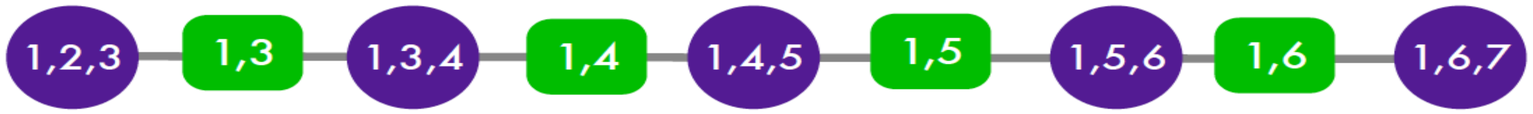
Junction Tree -- Triangulation



- Clique size increases
- Separator set size increases



Junction Tree–Message Passing



- Joint Probability

$$p(x) \propto \psi(x_1, x_2, x_3)\psi(x_1, x_3, x_4)\psi(x_1, x_4, x_5)\psi(x_1, x_5, x_6)\psi(x_1, x_6, x_7)$$

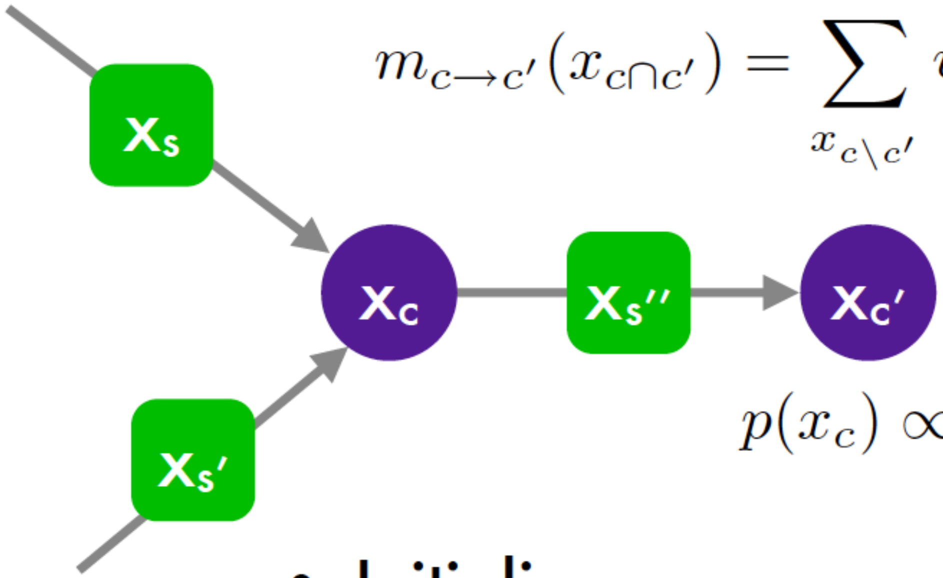
- Computing the normalization

$$m_{\rightarrow}(x_1, x_3) = \sum_{x_2} \psi(x_1, x_2, x_3)$$

$$m_{\rightarrow}(x_1, x_4) = \sum_{x_3} m_{\rightarrow}(x_1, x_3)\psi(x_1, x_3, x_4)$$

$$m_{\rightarrow}(x_1, x_5) = \sum_{x_4} m_{\rightarrow}(x_1, x_4)\psi(x_1, x_4, x_5)$$

Junction Tree−Message Passing（Sum-Production Algorithm）



$$m_{c\to c'}(x_{c\cap c'}) = \sum_{x_{c\setminus c'}} \psi_c(x_c) \prod_{c''\in N(c)\setminus c'} m_{c''\to c}(x_{c\cap c''})$$

$$p(x_c) \propto \psi_c(x_c) \prod_{c''\in N(c)} m_{c''\to c}(x_{c\cap c''})$$

- Initialize messages with 1
- Guaranteed to converge for (junction) trees
- Works well in practice even for loopy graphs
- Only local computations are required

Summary of the Junction Tree Algorithm

1. *Moralize* the graph
2. *Triangulate* the graph
3. *Propagate* by local message-passing in the junction tree

- Note that the first two steps are "off-line"
- Note also that these steps provide a bound of the complexity of the propagation step

Junction Tree Algorithm VS. (loopy) Belief Propagation

- There is an algorithm for exact inference on directed graphs without loops known as *belief propagation* (Pearl, 1988; Lauritzen and Spiegelhalter, 1988), and is equivalent to a special case of the sum-product algorithm. Here we shall consider only the sum-product algorithm because it is simpler to derive and to apply, as well as being more general.

- Here we consider one simple approach to approximate inference in graphs with loops, which builds directly on the previous discussion of exact inference in trees. The idea is simply to apply the sum-product algorithm even though there is no guarantee that it will yield good results. This approach is known as *loopy belief propagation* (Frey and MacKay, 1998) and is possible because the message passing rules (8.66) and (8.69) for the sum-product algorithm are purely local. However, because the graph now has cycles, information can flow many times around the graph. For some models, the algorithm will converge, whereas for others it will not.