Inference in Graphical Models

Prof. Jie Tang

2015

Contents

1. Introduction
2. Exact Inference
   - Variable Elimination
   - Message Passing Algorithms
3. Approximate Inference
   - Loopy Belief Propagation
   - Variational Inference
   - Sampling Based Method
4. Conclusion

Contents

1. Introduction
2. Exact Inference
   - Variable Elimination
   - Message Passing Algorithms
3. Approximate Inference
   - Loopy Belief Propagation
   - Variational Inference
   - Sampling Based Method
4. Conclusion

Introduction

- We now have a compact representation of a particular probability distribution using graphical models.
  - Bayesian Networks:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i | pa_{X_i})$$

  - Markov Random Field

$$P(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{f_a \in \mathcal{F}} f_a(X_a)$$

Introduction

- However, our ultimate goal is to answer various queries about the underlying distribution.

Introduction

- However, our ultimate goal is to answer various queries about the underlying distribution.
- What are the queries we are most interested in?

Introduction

- However, our ultimate goal is to answer various queries about the underlying distribution.
- What are the queries we are most interested in?
  - Marginal probability : $P(x) = \sum_y p(x|y)p(y)$

Introduction

- However, our ultimate goal is to answer various queries about the underlying distribution.
- What are the queries we are most interested in?
  - Marginal probability : $P(x) = \sum_y p(x|y)p(y)$
    - Conditional probabilities can be obtained by: $P(y|x) = \frac{p(x,y)}{p(x)}$

Introduction

- However, our ultimate goal is to answer various queries about the underlying distribution.
- What are the queries we are most interested in?
  - Marginal probability : $P(x) = \sum_y p(x|y)p(y)$
    - Conditional probabilities can be obtained by: $P(y|x) = \frac{p(x,y)}{p(x)}$
  - Posterior distributions:
    - Some of the nodes in a graph are clamped to observed values (X)
    - Compute posterior distributions of one or more subsets of nodes (latent variables Z), $p(Z|x)$
  - Maximum a posterior probability: $\text{argmax}_{x \in \mathcal{X}} P(x|y)$

Introduction

- However, our ultimate goal is to answer various queries about the underlying distribution.
- What are the queries we are most interested in?
  - Marginal probability : $P(x) = \sum_y p(x|y)p(y)$
    - Conditional probabilities can be obtained by: $P(y|x) = \frac{p(x,y)}{p(x)}$
  - Posterior distributions:
    - Some of the nodes in a graph are clamped to observed values (X)
    - Compute posterior distributions of one or more subsets of nodes (latent variables Z), $p(Z|x)$
  - Maximum a posterior probability: $\text{argmax}_{x \in \mathcal{X}} P(x|y)$
- How to make fast inference in graphical models is the key idea of this lecture!

Complexity of General Inference

- BAD NEWS:
  - Computing $P(X|E=e)$ in a graphical model is NP-hard.

$$P(X|E=e) = \frac{P(X,e)}{P(e)} = \frac{P(X,e)}{\sum_X P(X,e)}$$

Complexity of General Inference

- BAD NEWS:
  - Computing $P(X|E=e)$ in a graphical model is NP-hard.

$$P(X|E=e) = \frac{P(X,e)}{P(e)} = \frac{P(X,e)}{\sum_X P(X,e)}$$

- However, this does not mean we cannot solve the inference task.
  - It implies that we cannot find a general procedure that works efficiently for arbitrary graphical models
  - For particular families of graphical models, we can have provably efficient procedures
  - Even for arbitrary graphical models, we can find approximate methods to tackle the inference task!

Contents

1. Introduction

2. Exact Inference
   - Variable Elimination
   - Message Passing Algorithms

3. Approximate Inference
   - Loopy Belief Propagation
   - Variational Inference
   - Sampling Based Method

4. Conclusion

Naive Enumeration



We want to compute $P(E = e)$.

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

- A naive way is to enumerate over an exponential number of terms.
- However this computation cost is prohibitive when the number of variables gets large.

Variable Elimination on Directed Chains



By the factorization of Bayesian networks,

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

Variable Elimination on Directed Chains



We can perform the innermost summation

$$P(E=e) = \sum_d \sum_c \sum_b P(c|b)P(d|c)P(E=e|d) \sum_a P(a)P(b|a)$$
$$= \sum_d \sum_c \sum_b P(c|b)P(d|c)P(E=e|d)m_a(b)$$

- $m_a(b)$: a denotes the variable we are summing over (eliminating), b denotes the remaining arguments of the new factor m
- The summation "eliminates" one variable from our summation argument at a "local cost".

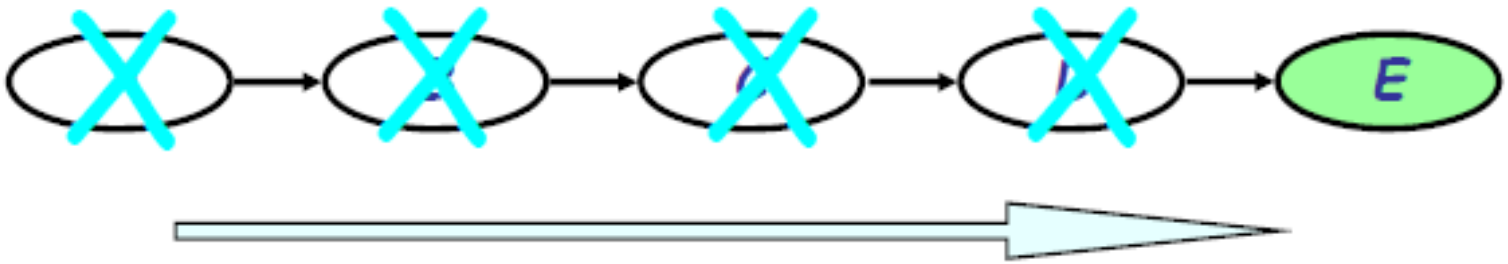
Variable Elimination on Directed Chains



Rearranging and then summing again, we get

$$
\begin{aligned}
P(E=e) &= \sum_d \sum_c \sum_b P(c|b)P(d|c)P(E=e|d)m_a(b) \\
&= \sum_d \sum_c P(d|c)P(E=e|d) \sum_b P(c|b)m_a(b) \\
&= \sum_d \sum_c P(d|c)P(E=e|d)m_b(c)
\end{aligned}
$$

Variable Elimination on Directed Chains



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

$$P(E = e) = \sum_{d} P(E = e|d)m_c(d)$$

Complexity Analysis

- We use n to denote the number of variables on the chain and k to denote the number of values a random variable can take.
- Each step costs $O(|val(X_i)|) \times O(|val(X_{i+1})|)$ operations: $O(nk^2)$
- Compare to naive evaluation that sums over joint values of $n-1$ variables: $O(n^k)$

Variable Elimination on Undirected Chains



Similarly, we have

$$
\begin{aligned}
P(E=e) &= \sum_d \sum_c \sum_b \sum_a P(a,b,c,d,e) \\
&= \sum_d \sum_c \sum_b \sum_a \frac{1}{Z}\phi(b,a)\phi(c,b)\phi(d,c)\phi(e,d) \\
&= \frac{1}{Z}\sum_d \sum_c \sum_b \phi(c,b)\phi(d,c)\phi(e,d)\sum_a \phi(b,a) \\
&= \frac{1}{Z}\sum_d \sum_c \phi(d,c)\phi(e,d)\sum_b \phi(c,b)m_a(b) \\
&= \dots
\end{aligned}
$$

Variable Elimination on Undirected Chains

- Marginal probability for middle a node

$$P(c) = \frac{1}{Z} \underbrace{\Big[\sum_a \phi(b,a)\Big[\sum_b \phi(c,b)\Big]\Big]}_{\alpha(c)} \cdot \underbrace{\Big[\sum_d \phi(d,c)\Big[\sum_e \phi(e,d)\Big]\Big]}_{\beta(c)}$$

- Interpretation as message passing

$$P(c) = \frac{1}{Z}\alpha(c) \cdot \beta(c)$$

Sum Product Operation

$$p \sum_{z} \prod_{\phi \in \mathcal{F}} \phi$$

- z: the variable to be eliminated
- ϕ: the factors associated with z
- $\mathcal{F}$: the set of factors associated with z

For example:

$$P(e) = \frac{1}{Z} \sum_{d} \sum_{c} \phi(d,c)\phi(e,d) \sum_{b} \phi(c,b) m_a(b)$$

$$= \ldots$$

Dealing With Evidence

- How can we turn the conditional probability into a sum product operation?
- Introduce the evidence potential

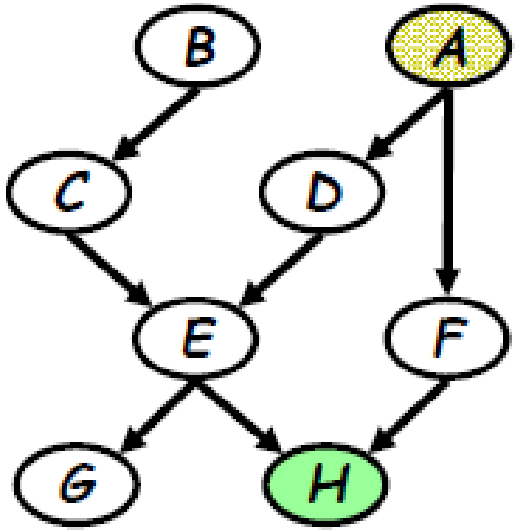
$$\delta(E, \tilde{e}) = 1 \quad \text{if and only if} \quad E = \tilde{e}.$$

- Then $P(E = \tilde{e}|X) = \sum_e P(e|X)\delta(E, \tilde{e})$

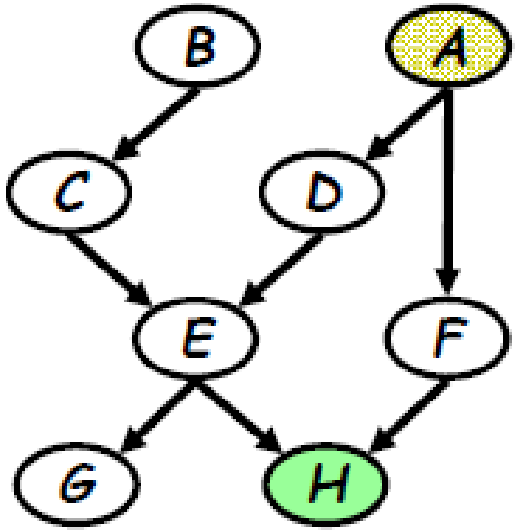
Now, we can build the variable elimination algorithm on the sum-product subroutine.
Let's see an example.

Variable Elimination on a Complex Network

- $\mathcal{X} = A, B, C, D, E, F, G, H$
- We went to compute $P(A|H = \tilde{h})$



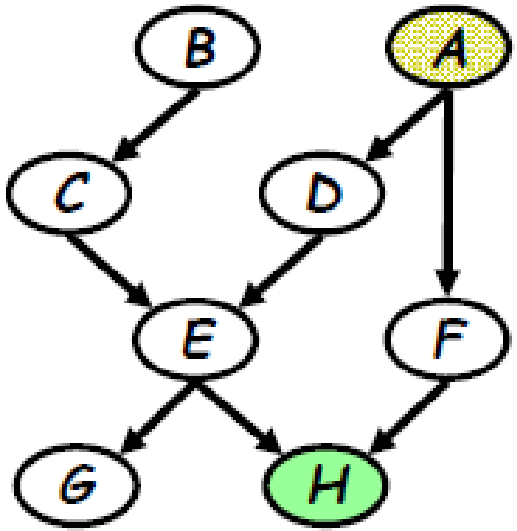
Initialize



Initialize$(G, \mathcal{I})$:

- Initialize $\mathcal{F}$ with the full set of factors, i.e. $\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c,d), P(f|a), P(g|e), P(h|e,f)\}$
- Choose an elimination order: $\mathcal{I} = H, G, F, E, D, C, B$

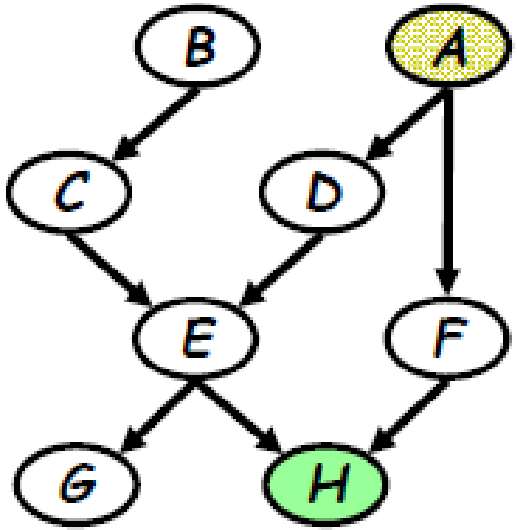
Evidence



Evidence(E):

- add the evidence potential $\delta(H, \tilde{h})$ to $\mathcal{F}$
  i.e. $\mathcal{F}$.append($\delta(H, \tilde{h})$)

Sum-Product-Variable-Elimination

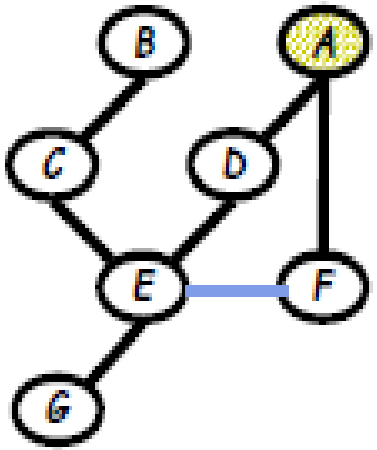


Sum-Product-Variable-Elimination($\mathcal{F}, \mathcal{I}$):

- $$\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c,d), P(f|a), P(g|e), P(h|e,f), \delta(H, \tilde{h})\}$$
- $\mathcal{I} = H, G, F, E, D, C, B$

Sum-Product-Variable-Elimination

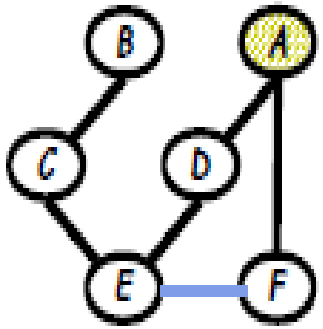
Eliminate H (Conditioning H):



$m_h(e, f) = \sum_h p(h|e, f)\delta(h, \tilde{h})$
$\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c, d), P(f|a), P(g|e), m_h(e, f)\}$

Sum-Product-Variable-Elimination

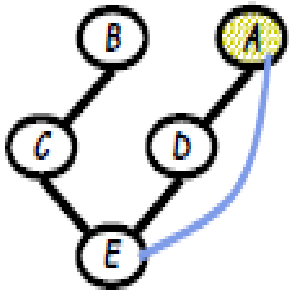
Eliminate G:



$m_g(e) = \sum_g P(g|e) = 1$
$\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c,d), P(f|a), m_h(e,f)\}$

Sum-Product-Variable-Elimination

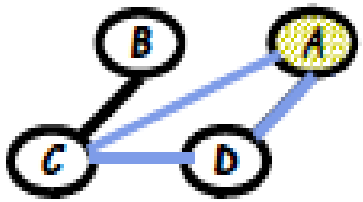
Eliminate F:



$m_f(e,a) = \sum_f P(f|a)m_h(e,f)$
$\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c,d), m_f(e,a)\}$

Sum-Product-Variable-Elimination

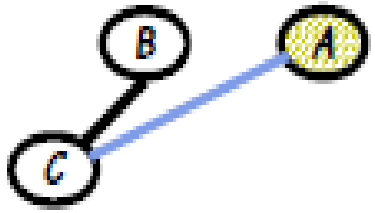
Eliminate E:



$m_e(a, c, d) = \sum_e P(e|c, d)m_f(e, a)$
$\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), m_e(a, c, d)\}$

Sum-Product-Variable-Elimination

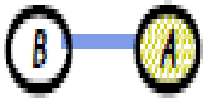
Eliminate D:



$m_d(a,c) = \sum_d P(d|a)m_e(a,c,d)$
$\mathcal{F} = \{P(a), P(b), P(c|b), m_d(a,c)\}$

Sum-Product-Variable-Elimination

Eliminate C:



$m_c(a,c) = \sum_c P(c|b)m_d(a,c)$
$\mathcal{F} = \{P(a), P(b), m_c(a,c)\}$

Sum-Product-Variable-Elimination

Eliminate B:



$m_b(a) = \sum_b P(b) m_c(a, b)$
$\mathcal{F} = \{P(a), m_b(a)\}$

Sum-Product-Variable-Elimination

- Conditioning H: $m_h(e,f) = \sum_h p(h|e,f)\delta(h,\tilde{h})$
  $\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c,d), P(f|a), P(g|e), m_h(e,f)\}$
- Eliminate G: $m_g(e) = \sum_g P(g|e) = 1$
  $\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c,d), P(f|a), m_h(e,f)\}$
- Eliminate F: $m_f(e,a) = \sum_f P(f|a)m_h(e,f)$
  $\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), P(e|c,d), m_f(e,a)\}$
- Eliminate E: $m_e(a,c,d) = \sum_e P(e|c,d)m_f(e,a)$
  $\mathcal{F} = \{P(a), P(b), P(c|b), P(d|a), m_e(a,c,d)\}$
- Eliminate D: $m_d(a,c) = \sum_d P(d|a)m_e(a,c,d)$
  $\mathcal{F} = \{P(a), P(b), P(c|b), m_d(a,c)\}$
- Eliminate C: $m_c(a,c) = \sum_c P(c|b)m_d(a,c)$
  $\mathcal{F} = \{P(a), P(b), m_c(a,c)\}$
- Eliminate B: $m_b(a) = \sum_b P(b)m_c(a,b)$
  $\mathcal{F} = \{P(a), m_b(a)\}$

Normalization

- Normalization $P(a|h = \tilde{h}) = \frac{P(a)m_b(a)}{\sum_a P(a)m_b(a)}$

Variable Elimination Algorithm

- Initialize($G, \mathcal{I}$):
  - Initialize the factor set $\mathcal{F}$
  - Choose an elimination ordering (excluding the query node)
- Evidence(E):
  - Append all the evidence potential to the factor set $\mathcal{F}$
- Sum-Product-Variable-Elimination($\mathcal{F}, \mathcal{I}$):
  - Iteratively sum-product a variable according to the elimination ordering
- Normalization

Complexity Analysis of Variable Elimination

For each variable x to be eliminated, the elimination procedure can be broken down into two parts:

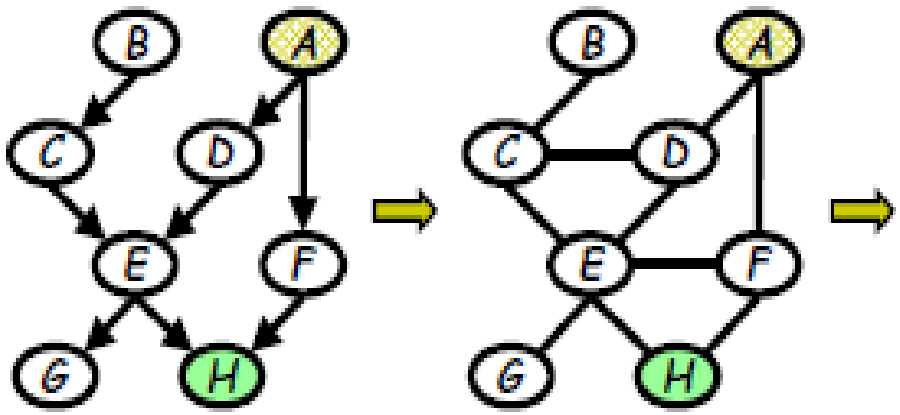
- Product: $m'_x(x, y_1, \ldots, y_k) = \prod_i^k m_i(x, y_{c_i})$
- Sum: $m_x(y_1, \ldots, y_k) = \sum_x m'_x(x, y_1, \ldots, y_k)$
- Multiplication: $k \times |val(X)| \times \prod_i |val(Y_{C_i})|$
- Addition: $|val(X)| \times \prod_i |val(Y_{C_i})|$

The major benefit of variable elimination is to reduce a globally exponential task to one which is locally exponential but globally polynomial.

Graph Elimination

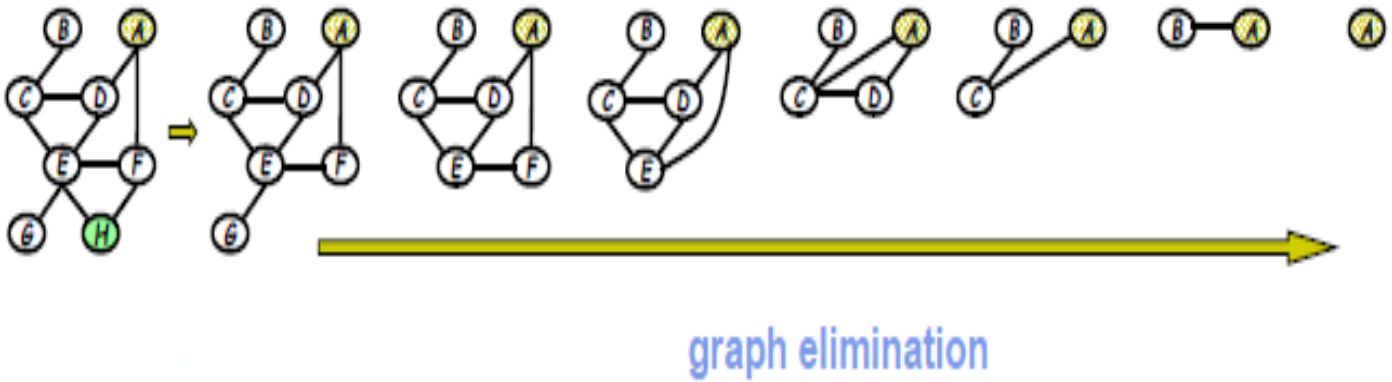
- For an undirected graphical model or a moralized Bayesian network
- Given graph $G(V, E)$, and an elimination ordering I
- For each variable in the ordering
  - Connect all neighbors of the variable
  - Eliminate the variable

Moralization



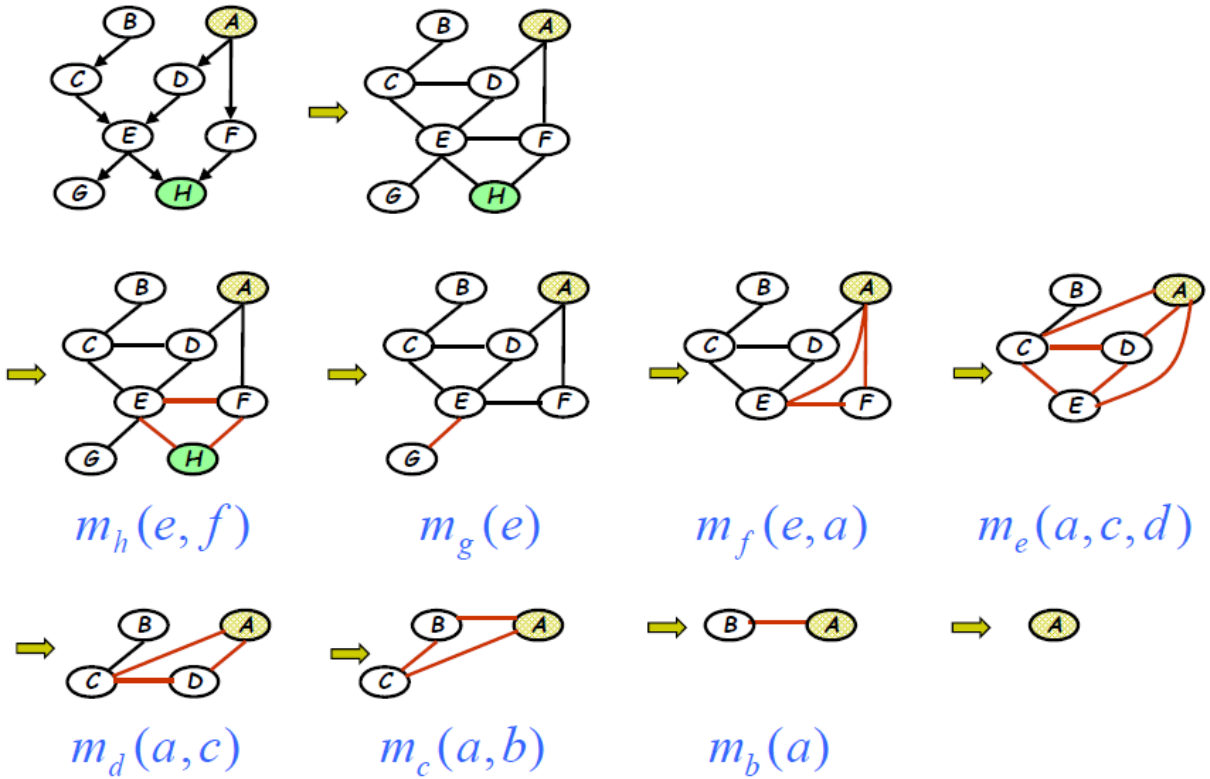
- Connecting nodes that have a common child (Marrying the parents)
- Making all edges in the graph undirected.

Graph Elimination



- Given graph $G(V, E)$, and an elimination ordering $I = H, G, F, E, D, C, B$
- For each variable in the ordpering
  - Connect all neighbors of the variable
  - Eliminate the variable

Elimination Clique

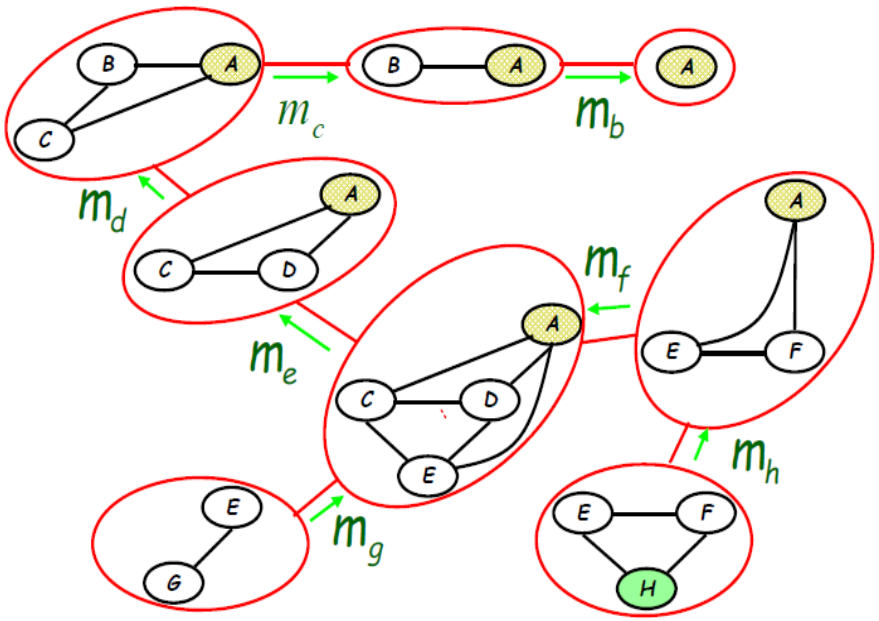


Variable Elimination and Graph Elimination:

- Intermediate terms in variable elimination correspond to the cliques resulted from graph elimination.

Limitations of Variable Elimination

We need to do one variable elimination procedure for each query. What if we have many queries to answer?



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

And this leads to message passing algorithms.

Contents

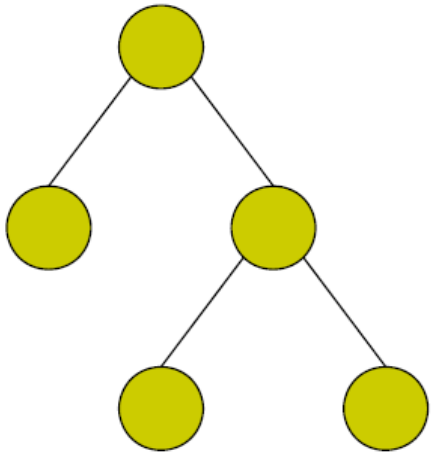
1. Introduction

2. Exact Inference
   - Variable Elimination
   - Message Passing Algorithms

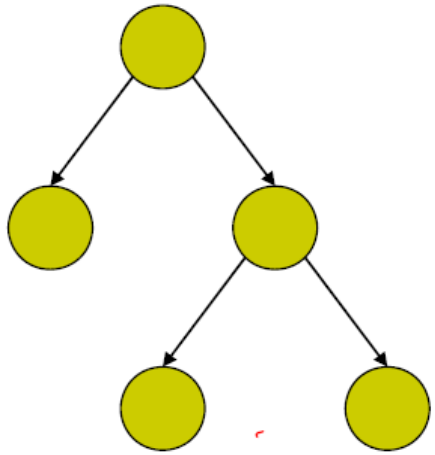
3. Approximate Inference
   - Loopy Belief Propagation
   - Variational Inference
   - Sampling Based Method

4. Conclusion

Tree Graphical Models



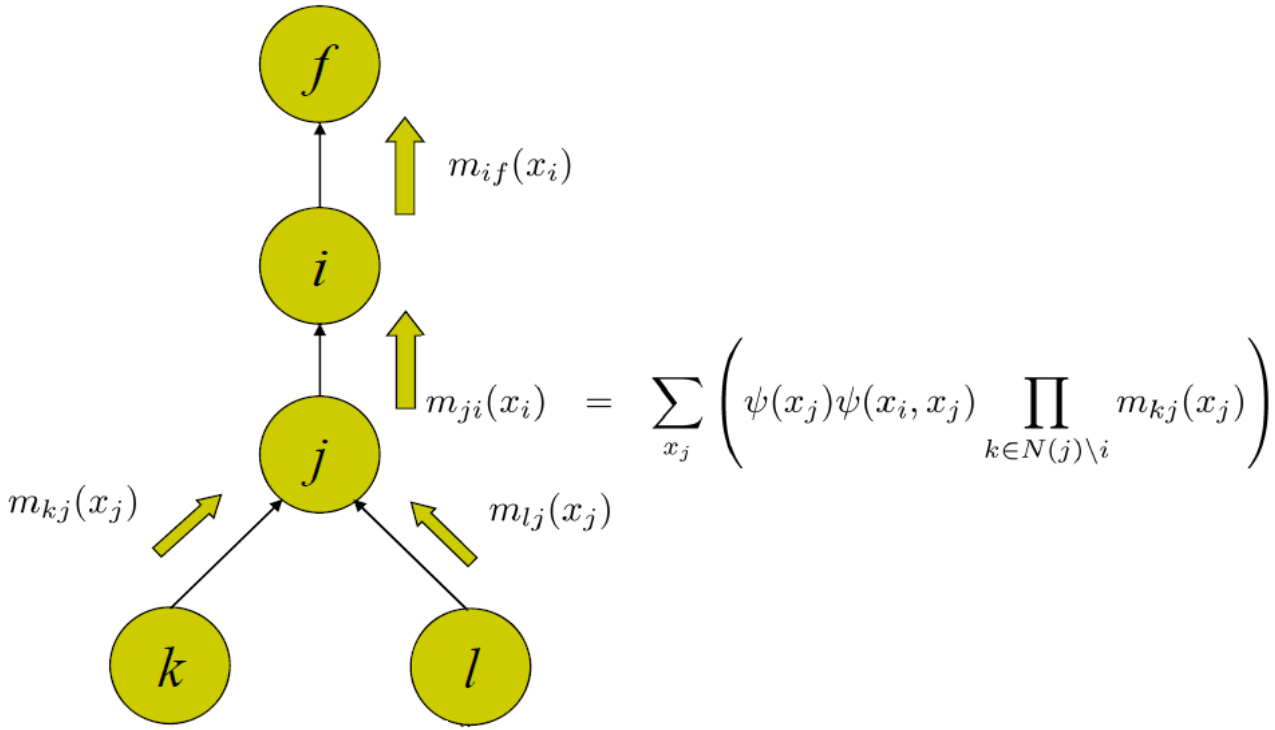
Undirected tree: a unique path between any pair of nodes



Directed tree: all nodes except the root have exactly one parent

Equivalence between Elimination and Message Passing

We can view variable elimination as a message passing process from leaf nodes to the root node on a tree.

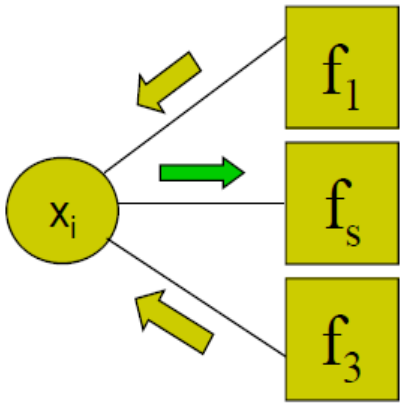


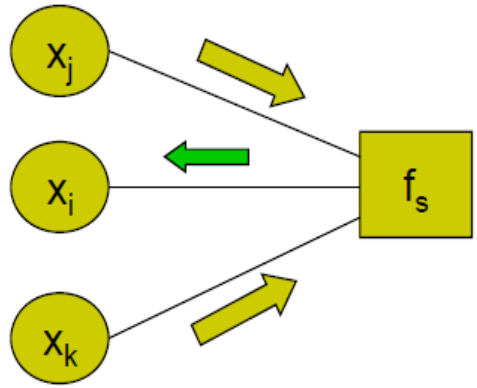
$p(x_f) \propto \psi(x_f) \prod_{e \in N(f)} m_{ef}(x_f)$

the key protocol is that a node can only pass a message along after it has received messages from its other neighbors

Belief Propagation on Factor Graphs

- Factor Tree: A Factor graph is a Factor Tree if the undirected graph obtained by ignoring the distinction between variable nodes and factor nodes is an undirected tree
- Message Passing:
  - ν from variables to factors
  - μ from factors to variables



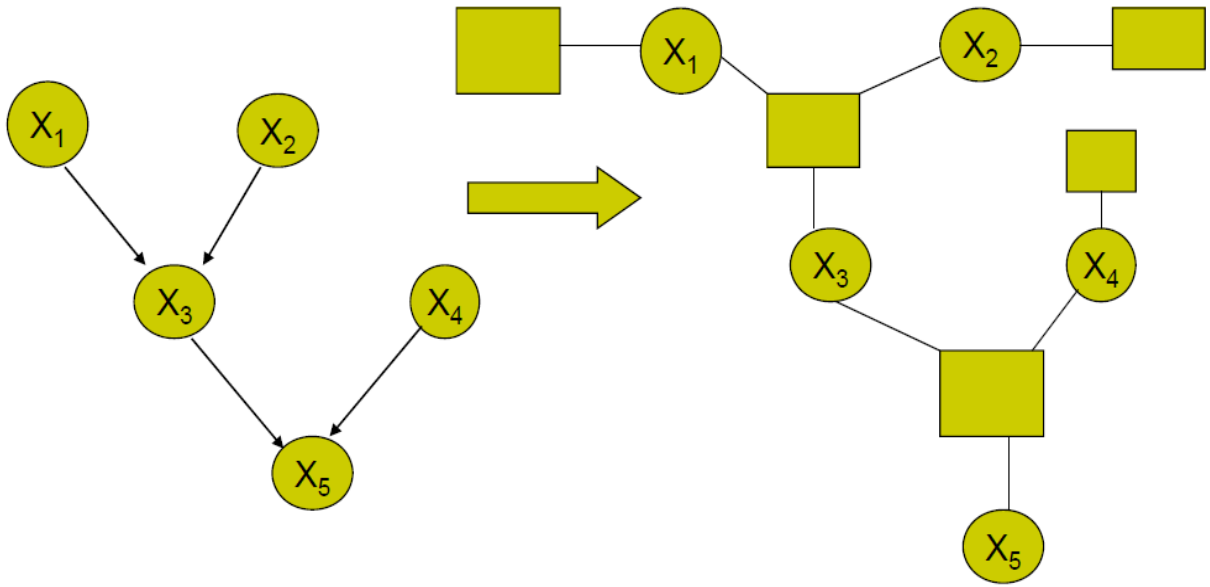


$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{ti}(x_i)$$

$$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s)} \setminus i} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(x_j) \right)$$

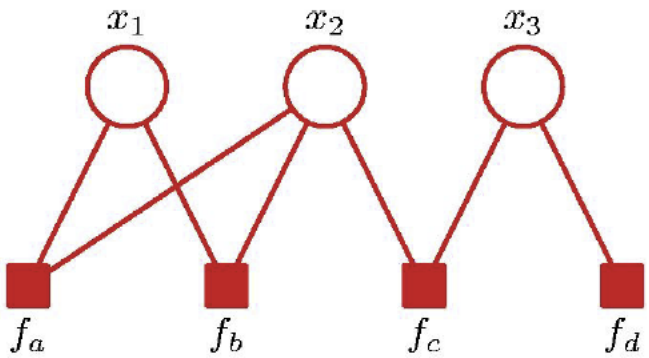
Why Factor Graph?

- Factor graph turns tree-like graphs to factor trees, including poly-trees.



- trees are a data-structure that guarantees correctness of belief propagation!

The Sum-product Algorithm for Factor Graphs



- Factorization

$$p(X) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

- Sum-product algorithm
  - assumption: all variables are discrete
  - first convert a tree structure graph into a factor graph
  - can deal with directed/undirected using the same framework
  - Belief propagation is a special case of sum-product algorithm

Overview of Sum-product Algorithm

- Assume that all variables are hidden
- Marginal is defined as

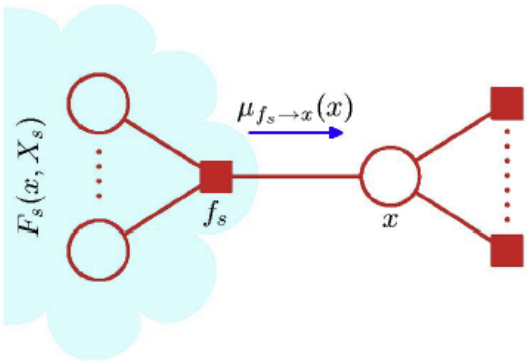
$$p(x) = \sum_{x' \in X \setminus x} p(x')$$

- Substitute for $p(x)$ the factor-graph expression

$$p(x) = \prod_{s} f_s(X_s)$$

- Interchange the summation and products for efficiency

Message over subtrees

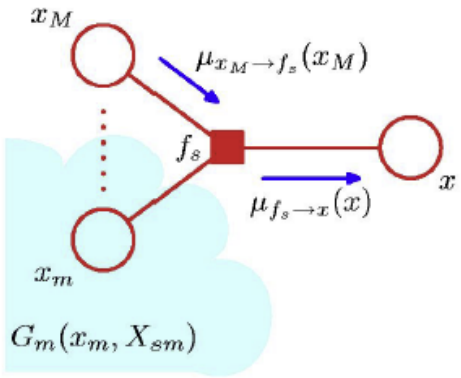


- Tree structure allows partitioning factors into groups

$$
\begin{aligned}
p(x) &= \prod_{s \in ne(x)} F_s(x, X_s) \\
&= \prod_{s \in ne(x)} \Big[\sum_{X_s} F_s(x, X_s) \Big] \\
&= \prod_{s \in ne(x)} \mu_{f_s \to x}(x)
\end{aligned}
$$

where $\mu_{f_s \to x}(x)$ is viewed as message sent from factor function f_s to variable node x; $p(x)$ is product of all incoming messages at node x.

Messages sent from variable to factor nodes



- Each $F_s(x, X_s)$ is described by a factor subgraph

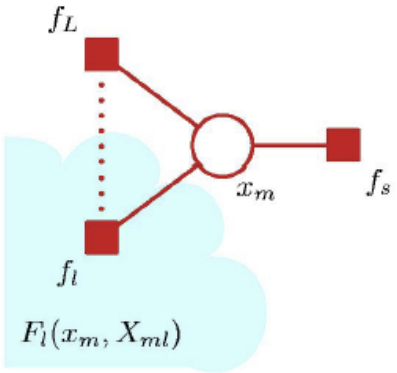
$$F_s(x, X_s) = f_s(x, x_1, \cdots, x_M) G_M(x_M, X_{sm})$$

- Re-substituting

$$\mu_{f_s \to x}(x) = \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \cdots, x_M) \prod_{m \in ne(f_s) \setminus x} \underbrace{\left[\sum_{X_{sm}} G_m(x_m, X_{sm}) \right]}_{\mu_{x_m \to f_s}(x_m)}$$

- three steps:
  - take product of incoming messages into factor node
  - multiply by factor associated with the node
  - marginalize over all variables associated with incoming messages

Messages sent from variable to factor nodes



- Making use of sub-graph factorization

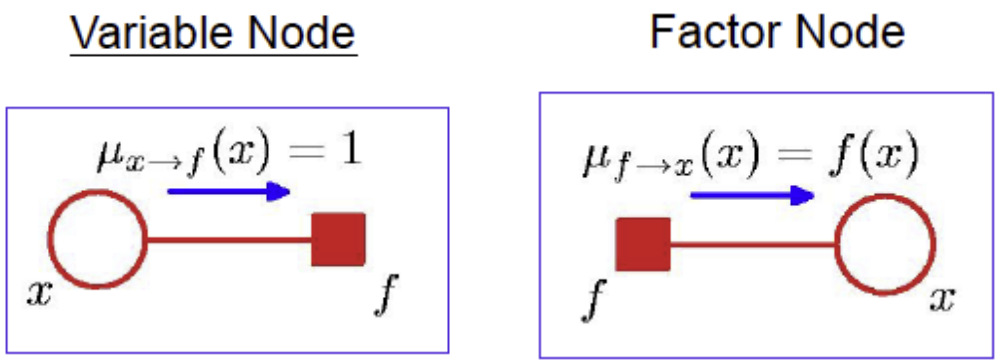
$$G_M(x_M, X_{sm}) = \prod_{l \in ne(x_m) \setminus f_s} F_l(x_m, X_{ml}))$$

- Substituting into expression for $\mu_{x_m \to f_s}(x_m)$

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

- Taking product of incoming messages along other links

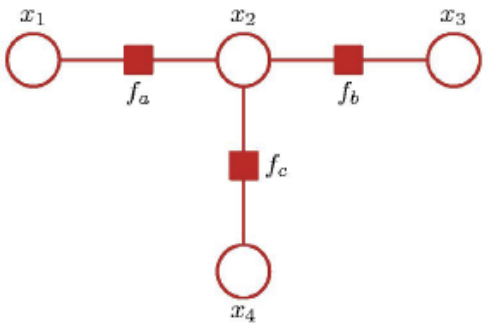
Sum-product Algorithm Messages



- Messages are initialized by
  - if leaf node x is a variable node, then it sends $\mu_{x\to f}(x) = 1$
  - if leaf node f is a factor node, then it sends $\mu_{f\to x}(x) = f(x)$
- Then messages passing are applied recursively
- Finally, marginal is calculated using

$$p(x) = \prod_{s\in ne(x)} \left[\sum_{X_s} F_s(x, X_s)\right] = \prod_{s\in ne(x)} \mu_{f_s\to x}(x)$$

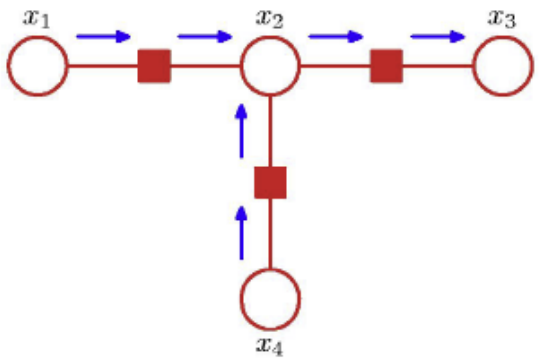
Finding Marginals for All Nodes



- Only need to run the algorithm for one time
- Pick up any variable (e.g., x_3) as root
- Perform message passing from leafs to root
- In the above example, the four-node factor graph has a joint distribution

$$p(x) = f_a(x_1, x_2) f_b(x_2, x_3) f_e(x_2, x_4)$$

Sum-product Algorithm Messages



Messages from leaf nodes

$$\mu_{x_1 \to f_a}(x_1) = 1$$

$$\mu_{f_a \to x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2)$$

$$\mu_{x_4 \to f_c}(x_4) = 1$$

$$\mu_{f_c \to x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4)$$

$$\mu_{x_2 \to f_b}(x_2) = \mu_{f_a \to x_2}(x_2)\mu_{f_c \to x_2}(x_2)$$

$$\mu_{f_b \to x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3)\mu_{x_2 \to f_b}$$

Messages from root node

$$\mu_{x_3 \to f_b}(x_3) = 1$$

$$\mu_{f_b \to x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3)$$

$$\mu_{x_2 \to f_a}(x_2) = \mu_{f_b \to x_2}(x_2)\mu_{f_b \to x_2}(x_2)$$

$$\mu_{f_b \to x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2)\mu_{x_2 \to f_a}(x_2)$$

$$\mu_{x_2 \to f_c}(x_2) = \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2)$$

$$\mu_{f_c \to x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4)\mu_{x_2 \to f_c}(x_2)$$

Complexity Analysis and Two-pass Algorithm

- We are interested in the complexity of computing all the node marginals.
- Naively this complexity of message passing is $O(NC)$ (where N=nodes, C=complexity of one complete passing/clique bottleneck).
- Alternative dynamic programming approach: $O(2C)$

Contents

1. Introduction

2. Exact Inference
   - Variable Elimination
   - Message Passing Algorithms

3. Approximate Inference
   - Loopy Belief Propagation
   - Variational Inference
   - Sampling Based Method

4. Conclusion

Loopy Belief Propagation

The general idea behined Loopy Belief Propagation (LBP) is to run Belief Propagation on a graph containing loops, despite the fact that the presence of loops does not guarantee convergence.

- Empirically, a good approximation is still achievable
  - Stop after fixed number of iterations
  - Stop when no significant change in beliefs
  - If solution is not oscillatory but converges, it usually is a good approximation
  - More hints: **Flood schedule** simultaneously passes a message across every link in both direction

Contents

1. Introduction
2. Exact Inference
   - Variable Elimination
   - Message Passing Algorithms
3. Approximate Inference
   - Loopy Belief Propagation
   - Variational Inference
   - Sampling Based Method
4. Conclusion

Bethe Approximation

- Let's call the actual distribution P:

$$P(X) = \frac{1}{Z} \prod_{f_a \in F} f_a(X_a)$$

- We want to find a distribution Q that approximates P, to measure the distance, we use the KL divergence

$$\begin{aligned} KL(Q||P) &= \sum_X Q(X) \log \frac{Q(X)}{P(X)} \\ &= \sum_X Q(X) \log Q(X) - \sum_X Q(X) \log P(X) \\ &= -H_Q(X) - E_Q \log P(X) \end{aligned}$$

Bethe Approximation

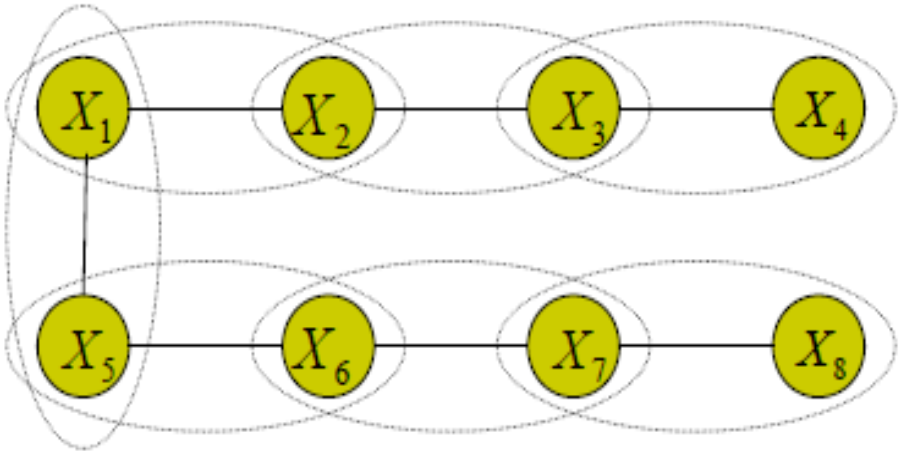
- Replace $P(X)$ with the definition mentioned above,

$$\begin{aligned} KL(Q||P) &= -H_Q(X) - E_Q \log(\frac{1}{Z} \prod_{f_a \in F} f_a(X_a)) \\ &= -H_Q(X) - \log \frac{1}{Z} - \sum_{f_a \in F} E_Q \log f_a(X_a) \\ &= -H_Q(X) - \sum_{f_a \in F} E_Q \log f_a(X_a) + \log Z \end{aligned}$$

- We can define a separate quantity "Gibbs Free Energy" for the first two terms in the last equation

$$F(P, Q) = -H_Q(X) - \sum_{f_a \in F} E_Q \log f_a(X_a)$$

Tree Energy Function



- The probability distribution of a Factor Tree can be factorized as

$$b(x) = \prod_a b_a(x_a) \prod_i b_i(x_i)^{1-d_i}$$

where the first product term is over the doubleton factors, the second product term is over the singleton factors and d_i is the number of direct neighbors of a singleton node X_i.

Tree Energy Function

- Entropy:

$$H_{tree} = -\sum_{a}\sum_{x_a} b_a(x_a)\log b_a(x_a) + \sum_{i}(d_i - 1)\sum_{x_i} b_i(x_i)\log b_i(x_i)$$

- Gibbs Free Energy

$$F_{tree} = \sum_{a}\sum_{x_a} b_a(x_a)\log\frac{b_a(x_a)}{f_a(x_a)} + \sum_{i}(d_i - 1)\sum_{x_i} b_i(x_i)\log b_i(x_i)$$

Bethe Approximation

We want to use a simple form to approximate the Gibbs free energy for any general factor graph.

- Bethe Approximation

$$H_{Bethe} = -\sum_a \sum_{x_a} b_a(x_a) \log b_a(x_a) + \sum_i (d_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)$$

$$F_{Bethe} = \sum_a \sum_{x_a} b_a(x_a) \log \frac{b_a(x_a)}{f_a(x_a)} + \sum_i (1 - d_i) \sum_{x_i} b_i(x_i) \log b_i(x_i)$$

- The Bethe free energy F_{Bethe} is equal to the Gibbs free energy $F(P, Q)$ for tree-structured graphs, but for general graphs it is not the case.

Constrained Optimization

$$\begin{aligned} \text{min} \quad & F_{Bethe} \\ \text{subject to} \quad & \sum_{x_i} b_i(x_i) = 1, \text{ for all } i \\ & b_i(x_i) = \sum_{X_a \backslash x_i} b_a(X_a), \text{for all } a \text{ and } i \end{aligned}$$

Thus, the Lagragian dual form is

$$\begin{aligned} L = F_{Bethe} &+ \sum_i \gamma_i \{1 - \sum_{x_i} b_i(x_i)\} \\ &+ \sum_a \sum_{i \in N(a)} \sum_{x_i} \lambda_{ai}(x_i) \{b_i(x_i) - \sum_{X_a \backslash x_i} b_a(X_a)\} \end{aligned}$$

Constrained Optimization

Setting the derivative with respect to the parameters to zero:

$$\frac{\partial L}{\partial b_i(x_i)} = 0 \Longrightarrow b_i(x_i) \propto \exp(\frac{1}{d_i - 1} \sum_{a \in N(i)} \lambda_{ai}(x_i))$$

$$\frac{\partial L}{\partial b_a(X_a)} = 0 \Longrightarrow b_a(X_a) \propto \exp(-\log f_a(X_a) + \sum_{i \in N(a)} \lambda_{ai}(x_i))$$

If we set $\lambda_{ai}(x_i) = \log m_{i \to a} = \log \prod_{b \in N(i)\ a} m_{b \to i}(x_i)$, we obtain

$$b_i(x_i) \propto f_i(x_i) \prod_{a \in N(i)} m_{a \to i}(x_i)$$

$$b_a(X_a) \propto f_a(X_a) + \prod_{i \in N(a)} \prod_{c \in N(i)\ a} m_{c \to i}(x_i)$$

Mean Field Approximation

- Given the actual distribution $p(X|\theta)$
- We want to use a fully factorized distribution $q(X|\theta)$ to approximate the original one, that is

$$q(x_1, \ldots, x_m) = \prod_i q_i(x_i)$$

- And this is called the mean field approximation.
- Similar to loopy belief propagation, we want to find a fixed-point characterization for the Gibbs free energy. $F(P; Q)$.

Optimization

$$\begin{aligned} \text{min} \quad & F_{P,Q} = -H_Q(X) + \sum_{f_a \in F} E_Q(\log f_a(x_a)) \\ \text{subject to} \quad & q(x_1, \ldots, x_m) = \prod_i q_i(x_i) \\ \text{and} \quad & \sum_{x_i} q_i(x_i) = 1, \text{ for all } i \end{aligned}$$

The two terms in the right hand side can be simplified as

$$H_Q(X) = \sum_i H_Q(X_i)$$

$$E_Q \log f_a(x_a) = \sum_{x_a} \Big(\prod_{X_i \in X_a} q_i(x_i) \Big) \log f_a(x_a)$$

Contents

1. Introduction

2. Exact Inference
   - Variable Elimination
   - Message Passing Algorithms

3. Approximate Inference
   - Loopy Belief Propagation
   - Variational Inference
   - Sampling Based Method

4. Conclusion

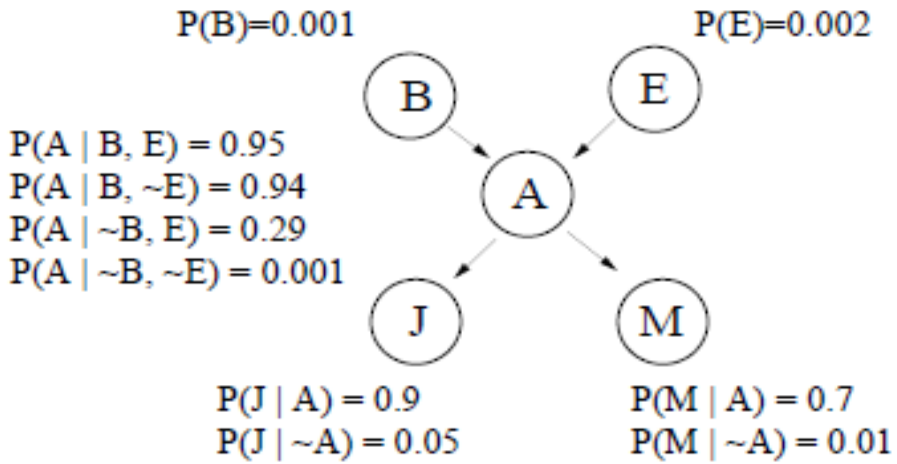
Sampling Based Method

For any distribution P and function f,

$$E_P[f] \approx \frac{1}{M} \sum_{m=1}^{M} f(x_m)$$

Hint: we only need to randomly sample from the underlying distribution given by the graphical model.
However, sampling is not easy at all.

Forward Sampling in Bayesian Networks



To generate a sample $X = (B, E, A, J, M)$:

- Sample $B \sim \text{Ber}(0.001)$: $r \sim U(0, 1)$. If $(r < 0.001)$ then $B = 1$ else $B = 0$.
- Sample $E \sim \text{Ber}(0.002)$
- If $B = 1$ and $E = 1$, sample $A \sim \text{Ber}(0.95)$, and so on
- If $A = 1$ sample $J \sim \text{Ber}(0.9)$ else $J \sim \text{Ber}(0.05)$
- If $A = 1$ sample $M \sim \text{Ber}(0.7)$ else $M \sim \text{Ber}(0.01)$

Querying without Evidence

- Goal: Estimate $P(X = x)$
  - Generate samples from the Bayesian network
  - Compute fraction where $X = x$

Querying with Evidence

- Goal: Estimate $P(X = x|E = e)$
- Rejection Sampling
  - Generate samples from the Bayesian network
  - Throw away all those where $E \neq e$
  - Compute fraction where $X = x$
- Expected fraction of samples kept $\sim P(E = e)$

Summary for Direct Sampling

- Generating samples from a Bayesian network is "easy"
- With evidence, number of required samples grows exponentially with number of observed variables
- Forward sampling generally infeasible for Markov random fields.
- We need another way to tackle the sampling problem.

Markov Chain Monte Carlo

- Basic Concepts
- Gibbs Sampling

Markov Chains

- A Markov Chain is a sequence of random variables $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ with the Markov Property

$$P(x^{(n)} = x | x^{(1)}, x^{(2)}, \ldots, x^{(n-1)}) = P(x^{(n)} = x | x^{(n-1)})$$

- The next state depends only on the preceding state. Recall HMMs!
- $P(x^{(n)} = x | x^{(n-1)})$ is known as the transition kernel. shorthand: $T(x'|x)$
- We study homogeneous Markov Chains, in which the transition kernel is fixed with time
- We define $x^{(t)}$ to be the t-th sample of all variables in a graphical model.

Stationary Distribution

- Probability distributions over states: $\pi^{(t)}(x)$ is a distribution over the state of the system x, at time t
- Thus, we have

$$\pi^{(t+1)}(x') = \sum_x \pi^{(t)}(x) T(x'|x)$$

- Stationary distributions: $\pi^{(t)}(x)$ is stationary if it does not change under the transition probability:

$$\pi(x') = \sum_x \pi(x) T(x'|x) \quad \text{for all} \quad x$$

Important Concepts

- Irreducible: an MC is irreducible if you can get from any state x to any other state x' with probability > 0 in a finite number of steps
  - there are no unreachable parts of the state space
- Aperiodic: an MC is aperiodic if you can return to any state x at any time
  - Periodic MCs have states that need 2 time steps to return to (cycles)
- Ergodic (or regular): an MC is ergodic if it is irreducible and aperiodic
- Ergodicity is important: it implies you can reach the stationary distribution $\pi_{st}(x)$, no matter the initial distribution $\pi^{(0)}(x)$
  - All good MCMC algorithms must satisfy ergodicity, so that you cannot initialize in a way that will never converge

Important Concepts

- Reversible (detailed balance): an MC is reversible if there exists a distribution $\pi(x)$ such that the detailed balance condition is satisfied:

$$\pi(x')T(x|x') = \pi(x)T(x'|x)$$

- Then, we have $\pi(x)$ is a stationary distribution of the MC! (Prove it by yourself!)
- Thm: If a Markov chain is regular and satisfies detailed balance with respect to $p(x)$, then $p(x)$ is its unique stationary distribution
- The point now is how to build a transition matrix w.r.t $p(x)$ that satisfies the properties above.

Gibbs Sampling

- Suppose the graphical model contains variables $x_1, \ldots, x_n$
- Initialize starting values for $x_1^{(0)}, \ldots, x_n^{(0)}$
- For $t = 0, 1, \ldots, T$
  - Sample $x_1^{(t+1)} \sim P(x_1 | x_2^{(t)}, x_3^{(t)}, \ldots, x_n^{(t)})$
  - Sample $x_2^{(t+1)} \sim P(x_2 | x_1^{(t+1)}, x_3^{(t)}, \ldots, x_n^{(t)})$
  - $\ldots$
  - Sample $x_n^{(t+1)} \sim P(x_n | x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_{n-1}^{(t+1)})$

Conclusion

- Exact Inference
  - Variable Elimination
  - Message Passing Algorithms
- Approximate Inference
  - Loopy Belief Propagation
  - Variational Inference
  - Markov Chain Monte Carlo