Inference in Graphical Models

José Miguel Hernández-Lobato

Department of Engineering, Cambridge University

April 9, 2013

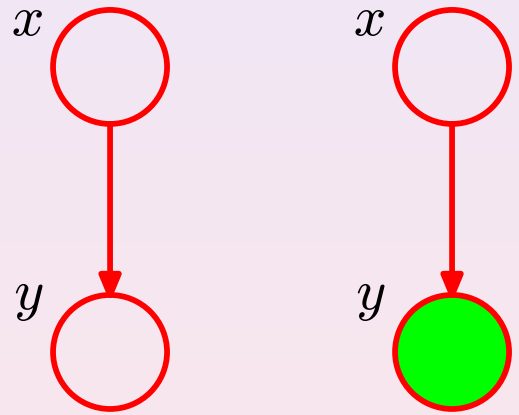
What is inference?

The problem of inference: Given that some of the variables in a GM are clamped to observed values, we want to compute the conditional distribution of a subset of other variables.

Simplest example: a Bayesian network (BN) for $p(x, y)$.

The BN represents the joint distribution for x and y as $p(x, y) = p(y|x)p(x)$.

Given y, we use Bayes' theorem to obtain $p(x|y) = p(y|x)p(x)/p(y)$, where $p(y) = \sum_x p(y|x)p(x)$.



Difficult problem with probably exponential cost in the worst case.

However, for some GM exact inference is tractable.

Variable Elimination

Given the BN $A \to B \to C \to D$ we want to compute $p(d)$.

The chain rule of BNs and the sum rule of probability theory lead to $p(d) = \sum_a \sum_b \sum_c p(d|c)p(c|b)p(b|a)p(a)$.

This operation has cost $\mathcal{O}(n^3)$, when A, B and C can take n values each.
However, we can reorder the computations to obtain
$p(d) = \sum_c p(d|c) \sum_b p(c|b) \sum_a p(b|a)p(a)$.

In this case, the computation of $p(d)$ has cost $\mathcal{O}(n^2)$.
Selecting a specific order of computations can produce large savings.

Main Idea

The GM expresses the joint distribution as a product of factors which depend only on a small number of variables.

Only need to compute some expressions once. By caching intermediate results, we avoid generating those expressions exponentially many times.

Sum-product Variable Elimination Algorithm

Procedure Sum-Product-VE

Input

Set of factors $\mathbf{\Phi}$.

Set of variables to be eliminated $\mathbf{Z}$.

Ordering of $\mathbf{Z}$: $Z_1, \ldots, Z_k$.

1: **for** $i = 1, \ldots, k$

2: $\quad \mathbf{\Phi} \leftarrow$ Eliminate-Var$(\mathbf{\Phi}, Z_i)$

3: **return** $\prod_{\phi \in \mathbf{\Phi}} \phi$

Procedure Eliminate-Var

Input

Set of factors $\mathbf{\Phi}$.

Variable to be summed out Z.

1: $\mathbf{\Phi}' \leftarrow \{\phi \in \mathbf{\Phi} : Z \in \text{Scope}[\phi]\}$

2: $\mathbf{\Phi}'' \leftarrow \mathbf{\Phi} - \mathbf{\Phi}'$

3: $\psi \leftarrow \prod_{\phi \in \mathbf{\Phi}'} \phi$

4: $\tau \leftarrow \sum_Z \psi$

5: **return** $\mathbf{\Phi}'' \cup \{\tau\}$

where scope(ϕ) is the set of variables on which ϕ is evaluated.

When a set of variables $\mathbf{Z}'$ are clamped to observed values, we

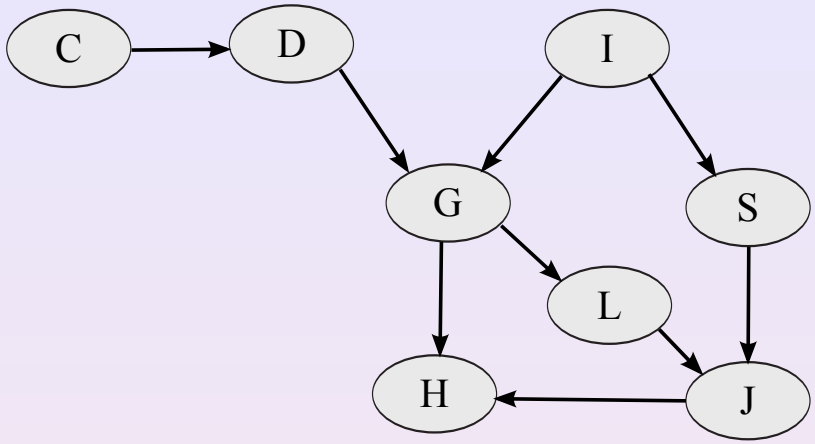
1 - Reduce the factors in $\mathbf{\Phi}$ which depend on any variable in $\mathbf{Z}'$.

2 - Eliminate only the variables in $\mathbf{Z} - \mathbf{Z}'$.

Example of the Execution of Variable Elimination

We ask for $p(J)$ for the joint distribution $p(C, D, I, H, G, S, L)$ given by the BN in the figure.

Slide source [Koller et al. 2009]



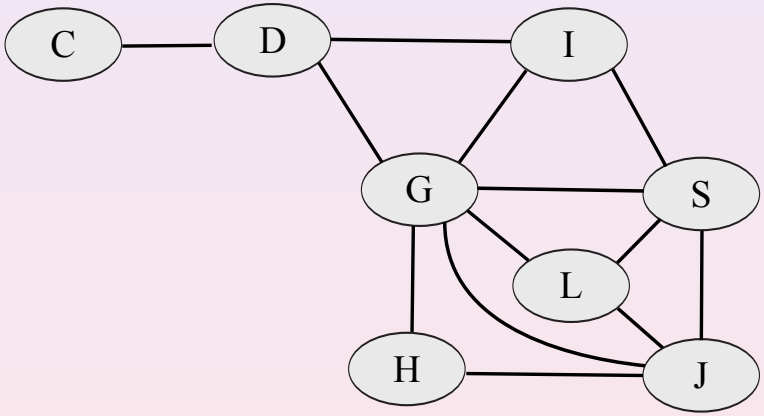
Step	Variable Eliminated	Factors used to compute ψ	Variables Involved	New Factor
1	C	$\phi_C(C)$, $\phi_D(D, C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G, I, D)$, $\tau_1(D)$	G I, D	$\tau_2(G, I)$
3	I	$\phi_I(I)$, $\phi_S(S, I)$, $\tau_2(G, I)$	G, S, I	$\tau_3(G, S)$
4	H	$\phi_H(H, G, J)$	H G, J	$\tau_4(G, J)$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

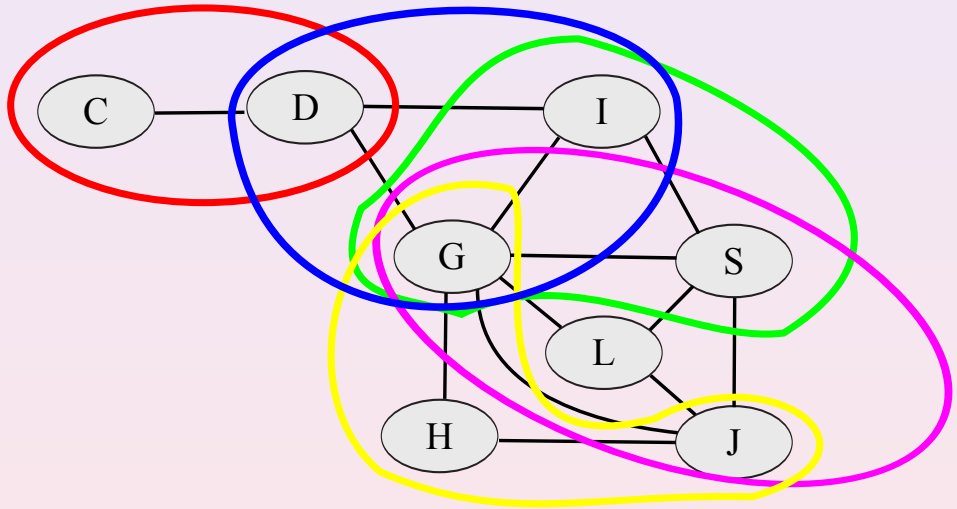
Induced Graph (IG)

Undirected graph where two random variables are connected if the two of them appear in some factor during the execution of variable elimination.

The scope of every factor produced during VE is a clique in the IG.

Every maximal clique is the scope of one of the ψ produced during VE.





The cost of VE is exponential in the number of nodes in the largest clique of the IG. Depends on the elimination ordering ! Finding the best ordering is NP-hard . Alternative: use a heuristic method .

Clique Tree

A run of VE defines a clique tree such that:

1 - We have a node for each factor ψ_i produced by VE.

2 - Each node is associated with the set of variables $\mathbf{C}_i = \text{Scope}[\psi_i]$.

3 - We connect $\mathbf{C}_i$ and $\mathbf{C}_j$ if τ_i is used to compute τ_j.

4 - For the link between $\mathbf{C}_i$ and $\mathbf{C}_j$ we have the sepset $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$.

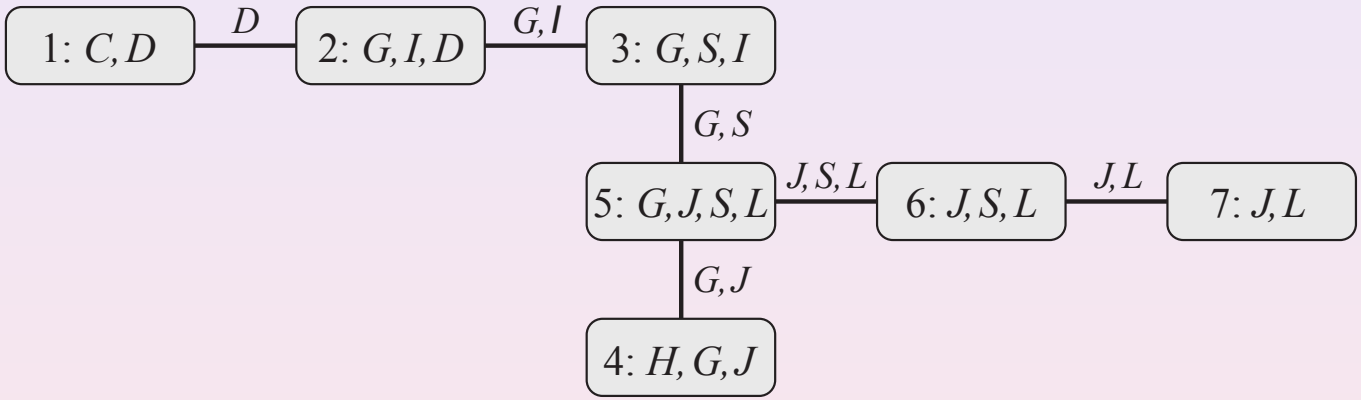


Figure [Koller et al. 2009].

Any clique tree must satisfy the following properties:

Running intersection property (RIP): for any variable X such that $X \in \mathbf{C}_i$ and $X \in \mathbf{C}_j$, then X is also in every node in the path between $\mathbf{C}_i$ and $\mathbf{C}_j$.

Family preservation property (FPP): each factor $\phi \in \mathbf{\Phi}$ is associated with a node $\mathbf{C}_i$ such that $\text{Scope}[\phi] \subseteq \mathbf{C}_i$.

Clique Tree Message Passing

VE can be implemented via message passing in a given clique tree.

By the FPP, any $\phi \in \mathbf{\Phi}$ is assigned to a node which we denote by $\alpha(\phi)$.

The initial potential for $\mathbf{C}_j$ is

$$\psi_j(\mathbf{C}_j) = \prod_{\phi:\alpha(\phi)=j} \phi\,.$$

The message from $\mathbf{C}_i$ to its neighbor $\mathbf{C}_j$ is

$$\delta_{i\to j} = \sum_{\mathbf{C}_i - \mathbf{S}_{ij}} \psi_i \prod_{k\in \mathrm{Nb}(i)-\{j\}} \delta_{k\to i}\,.$$

At any node i, we call *beliefs* the factor given by

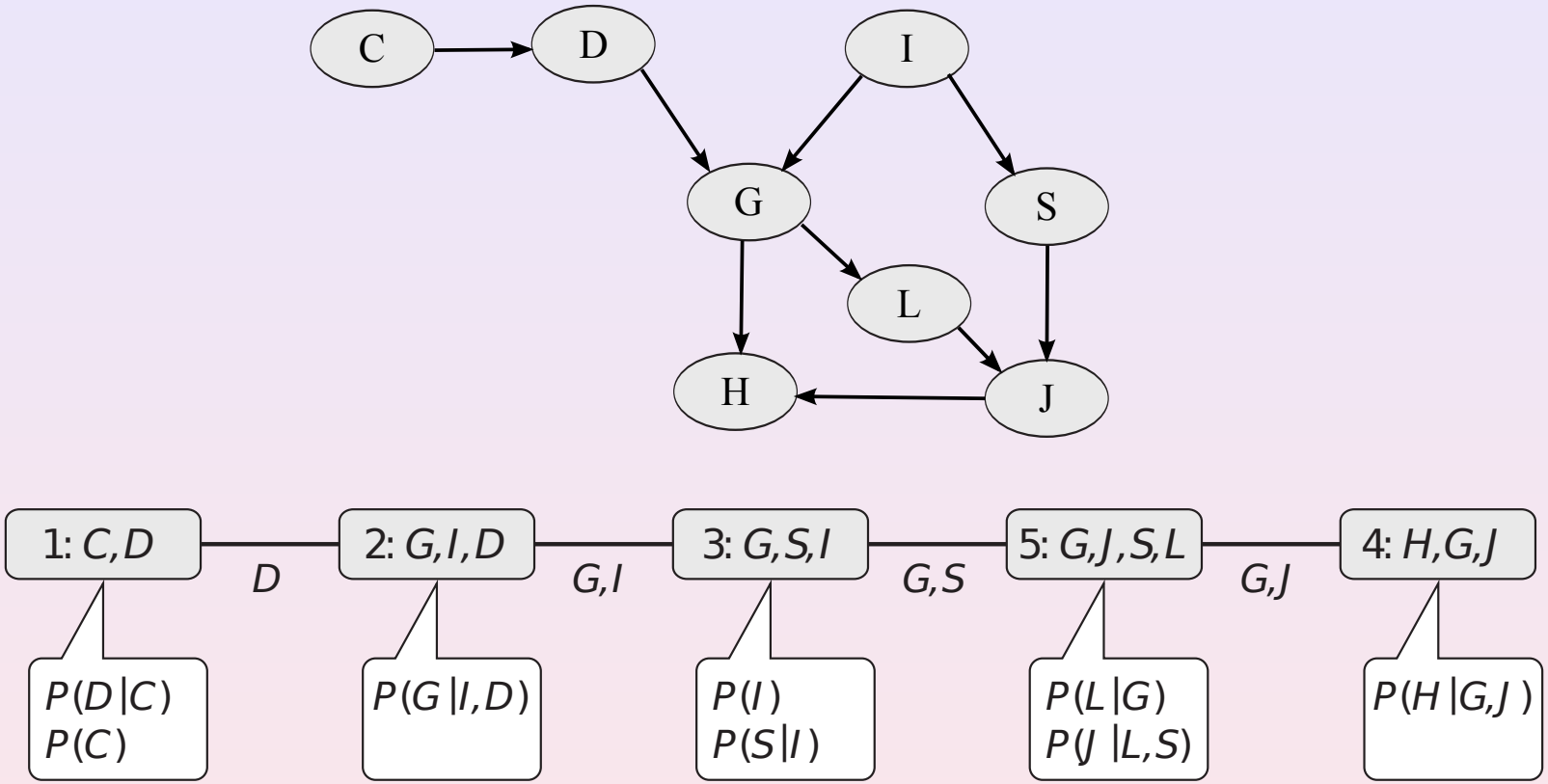
$$\beta_i(\mathbf{C}_i) = \prod_{j\in \mathrm{Nb}(i)} \delta_{j\to i} = \sum_{\cup_{j\neq i}\mathbf{C}_j} \prod_{\phi\in\mathbf{\Phi}} \phi\,.$$

To compute $p(Z_j)$ we follow the steps

- Select any $\mathbf{C}_i$ s.t. $Z_j \in \mathbf{C}_i$ and compute $\beta_i(\mathbf{C}_i)$ by message passing.
- Sum out in $\beta_i(\mathbf{C}_i)$ the variables $\mathbf{C}_i - \{Z_j\}$.

Example of Message Passing I

Given a GM we obtain a cluster tree that satisfies the RIP and the FPP.



To obtain $p(J)$ we select $\mathbf{C}_5$ and do message passing to get $\beta_5(G, J, S, L)$.

Figure source [Koller et al. 2009].

Example of Message Passing II

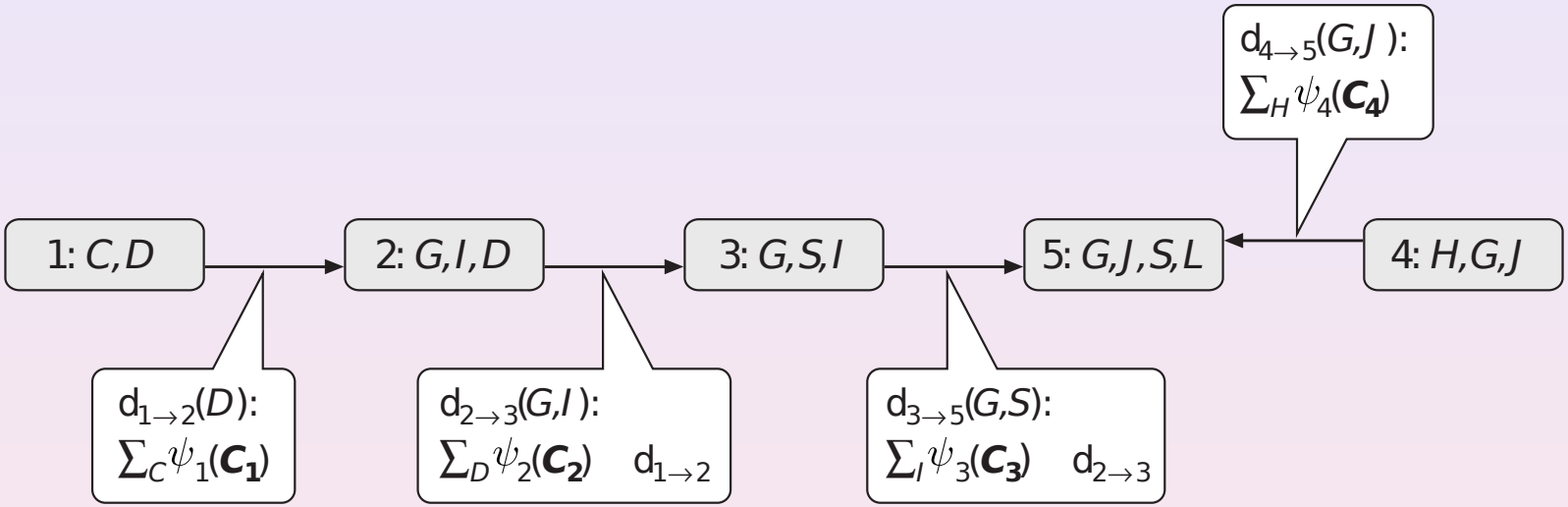


Figure source [Koller et al. 2009].

Calibration of a Clique Tree

Any edge $i \sim j$ in a CT has two messages associated $\delta_{i \to j}$ and $\delta_{j \to i}$.

To compute both messages for any edge and the beliefs for each node:

1 - Pick a random node as the root.
2 - Send all messages from the leaves to the root.
3 - Send all messages from the root to the leaves.
4 - Compute the beliefs for each node in the graph using the messages.

At the end, each node has the marginal over the variables in its scope.

We can compute all the marginals with only twice the cost of VE.

How to Construct a Clique Tree

By running Variable Elimination:

Any resulting $\mathbf{C}_i$ which is not a maximal clique in the IG is usually eliminated.

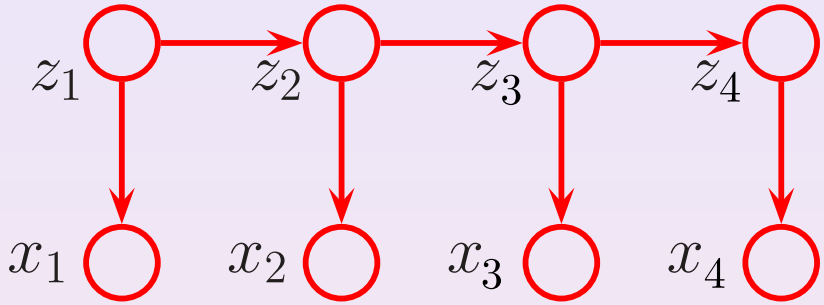
From a chordal graph (undirected graph with no loop larger than 3):

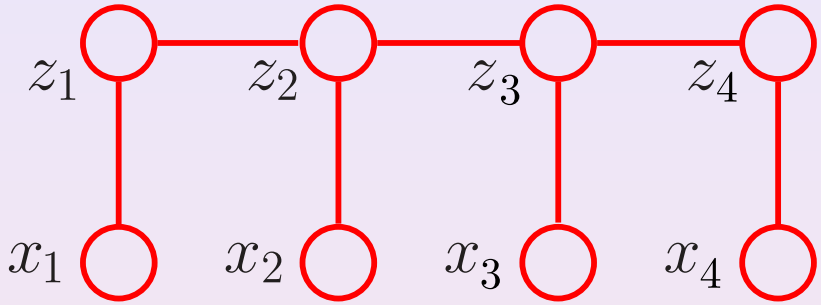
1 - Obtain an undirected graph by **moralization**.
2 - Obtain a chordal graph by **triangulation**.
3 - Find the **maximal cliques** in the chordal graph.
4 - Make each maximal clique a node in the clique tree.
5 - Find **maximum spanning tree** with weight $|\mathbf{C}_i \cap \mathbf{C}_j|$ for $i \sim j$.

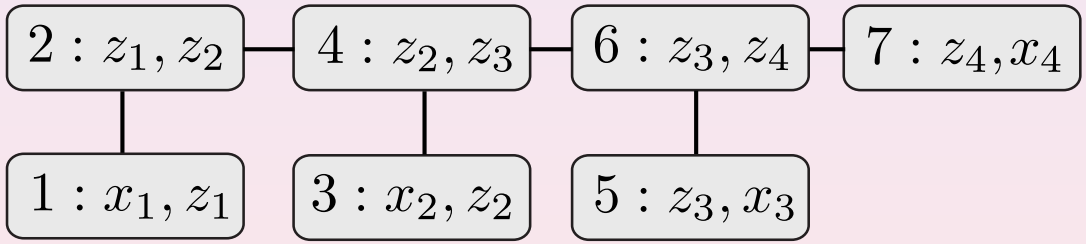
The computational cost of message passing is exponential in number of variables of the largest clique, exponential in the tree-width .

Finding a triangulation in which the largest clique has minimum size is NP-hard . Alternative: use a heuristic method .

Example: Clique Tree for a HMM







Approximate Inference

Exact inference is intractable if the tree-width of the clique tree is large.

What to do then?

Use approximate inference, trading off computational cost vs. accuracy.

We construct an approximation $\mathcal{Q}$ to the target distribution $\mathcal{P}$.

The approximation $\mathcal{Q}$ can be obtained by

1. Selecting a **simpler form** for $\mathcal{Q}$ that can be efficiently tuned to $\mathcal{P}$.
2. Drawing a **finite number of samples** from the distribution $\mathcal{P}$.
   $\mathcal{Q}$ is then built using these samples.

In the first case, approximate inference involves optimizing $\mathcal{Q}$ to match $\mathcal{P}$.

Some of these methods can be viewed as message passing on a graph.

Loopy Belief Propagation

We do message passing on a cluster graph rather than on a clique tree.

Cluster graph:

- Like a clique tree, but with cycles or loops.
- For the link between $\mathbf{C}_i$ and $\mathbf{C}_j$ we have the sepset $\mathbf{S}_{ij} \subseteq \mathbf{C}_i \cap \mathbf{C}_j$.

The cluster graph has to satisfy the **FPP** and a **generalized RIP**.

Running Intersection Property for Cluster Graphs

For any two nodes $\mathbf{C}_i$ and $\mathbf{C}_j$ containing variable X, there is precisely one path between them for which $X \in \mathbf{S}_e$ for all edges e in the path.

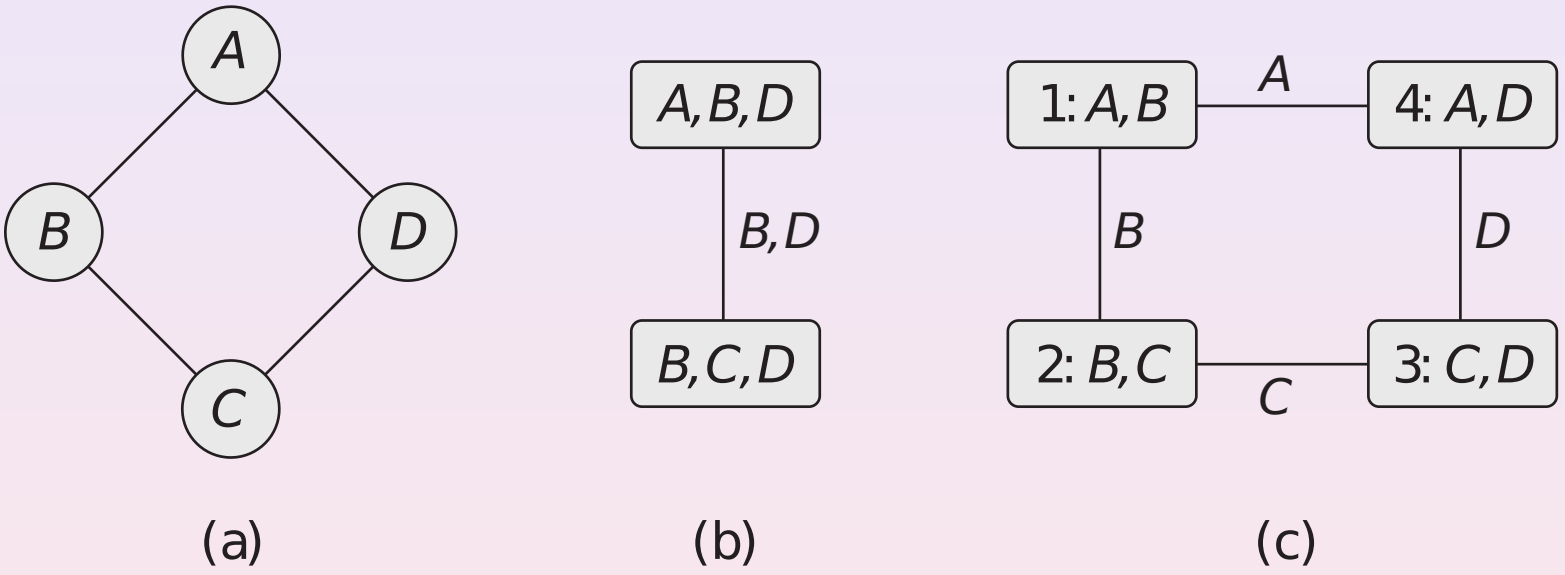
This implies that all edges associated with X form a tree that spans all the nodes containing X. Two nodes can be connected by multiple paths.

We can do inference on the cluster graph by message passing. However, because of the loops, we will often obtain only approximate answers.

Cluster Graph Example

Undirected graphical model with factors:

$\phi_1(A, B)$ $\phi_2(B, C)$ $\phi_3(C, D)$ $\phi_4(D, A)$



(a) (b) (c)

(a) - The undirected GM. (b) - A clique tree for the network in (a).
(c) - A cluster graph for the same network.

Figure source [Koller et al. 2009]

How to Choose the Cluster Graph?

When choosing the CG we have to consider a cost vs. accuracy trade-off.

Doing message passing in CGs that lead to more accurate results is more computationally expensive and vice-versa.

The chosen CG must also to satisfy the RIP and FPP.

A typical solution is the **Bethe cluster graph** (easily automated).

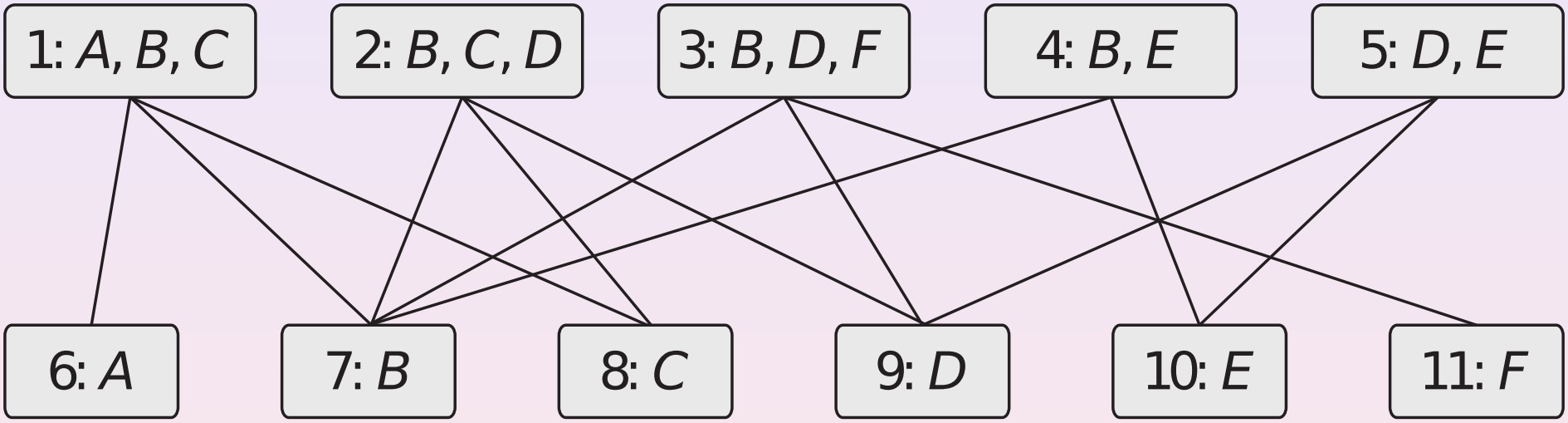
- A bipartite graph with two layers of large and small nodes.
- A large node $\mathbf{C}_{\alpha(\phi)}$ per factor ϕ in $\mathbf{\Phi}$, where $\mathbf{C}_{\alpha(\phi)} = \text{Scope}(\phi)$.
- A small node per random variable, with no associated factor.
- A large node $\mathbf{C}_i$ is connected to a small node $\mathbf{C}_j$ when $\mathbf{C}_j \subseteq \mathbf{C}_i$.

The Bethe cluster graph is guaranteed to satisfy both the RIP and FPP.

Example of Bethe Cluster Graph

Distribution with factors

$\phi_1(A, B, C)$ $\phi_2(B, C, D)$ $\phi_3(B, D, F)$ $\phi_4(B, E)$ $\phi_5(D, E)$



The Bethe cluster graph is limited in the sense that information between different clusters is passed through univariate marginal distributions.

Merging clusters can help capture interactions between multiple variables.

Loopy Belief Propagation Algorithm

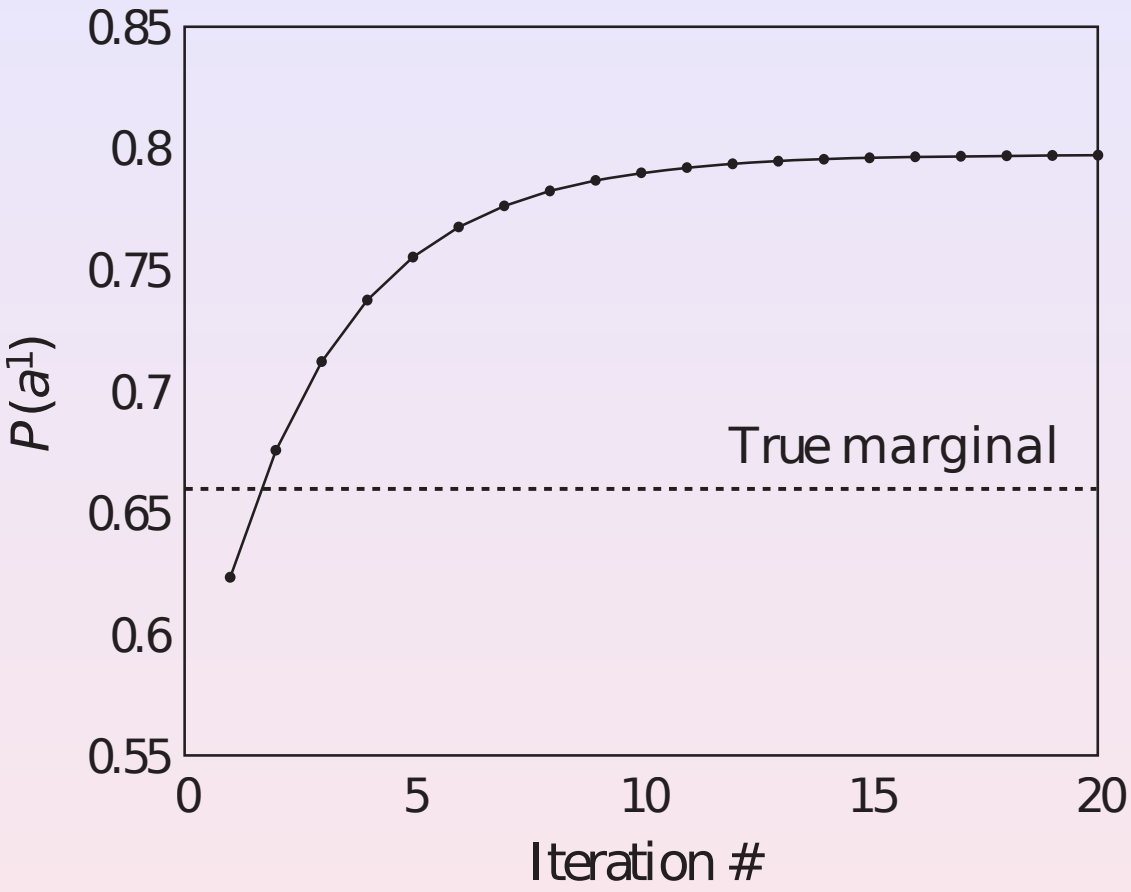
1. Assign each factor $\phi_k \in \Phi$ to a cluster $\mathbf{C}_{\alpha(k)}$.
2. Construct initial potentials $\psi_i(\mathbf{C}_i) = \prod_{k:\alpha(k)=i} \phi_k$.
3. Initialize all messages to be non-informative (*e.g.* equal to $\mathbf{1}$).
4. Repeat until convergence (*e.g.* messages no longer change)
   1. Select edge (i, j) and pass message:

$$\delta_{i \to j}(\mathbf{S}_{i,j}) = \sum_{C_i - S_{i,j}} \psi_i \times \prod_{k \in (\mathcal{N}_i - j)} \delta_{k \to i} \, .$$

5. Compute un-normalized beliefs $\beta_i(\mathbf{C}_i) = \psi_i \prod_{k \in \mathcal{N}_i} \delta_{k \to i}$.

At convergence, the marginals over the sepsets of adjacent nodes coincide and we have a calibrated cluster graph.

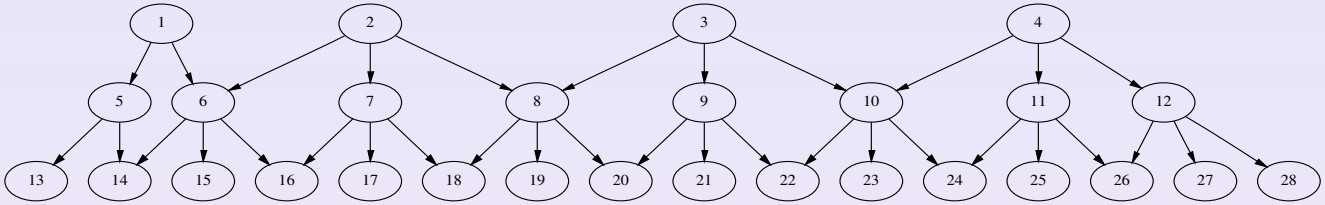
Example of Loopy Belief Propagation



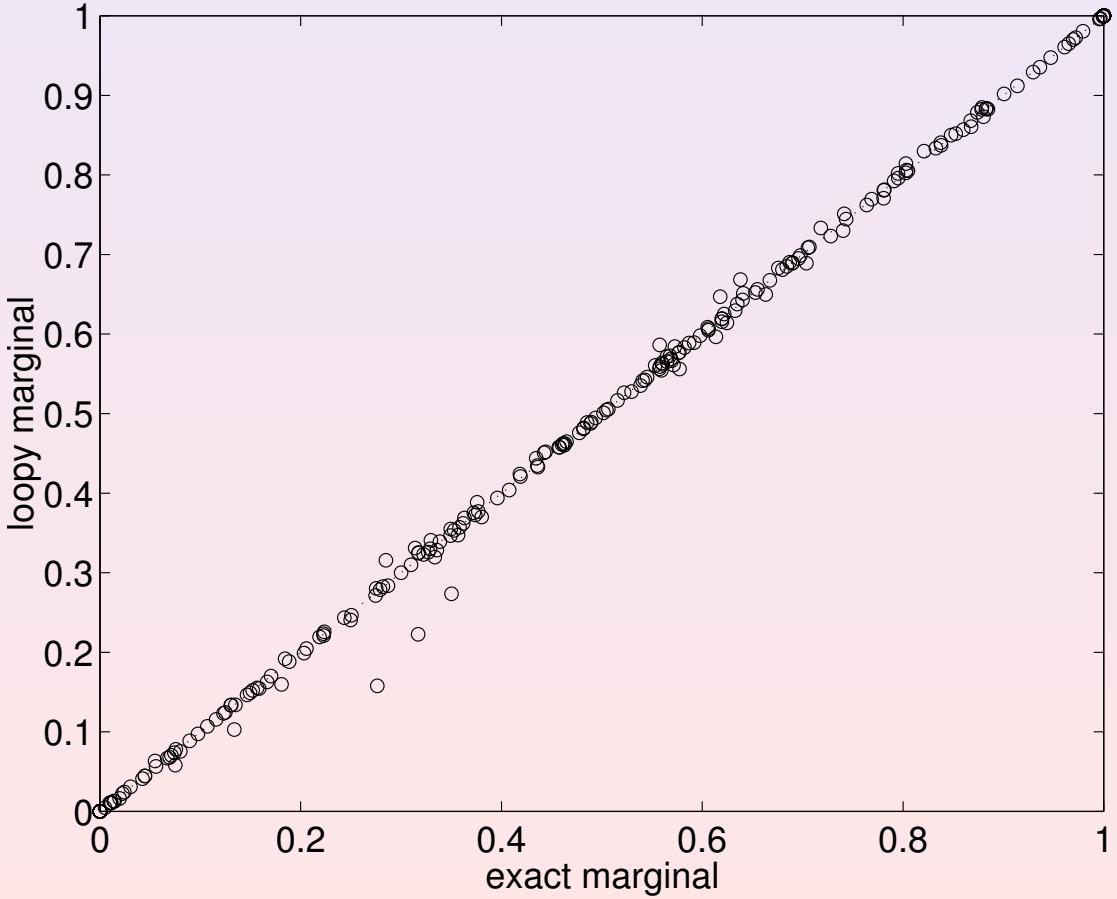
The algorithm converges but to a solution different from the exact one.

Figure source [Koller et al. 2009].

Loopy Belief Propagation: Accuracy of the Approximation



K. P. Murphy, Y. Weiss and M. I. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study, UAI 99



Summary

Inference in GMs is mainly the computation of conditional distributions.

Exact inference requires to sum an exponentially large joint distribution.

Variable elimination avoids the exponential blow up by caching intermediate results.

A clique tree allows us to do exact inference using message passing.
Message passing produces multiple answers in a single run.

When the tree-width of the clique tree is large we have to use approximations.

Loopy belief propagation allows us to do efficient approximate inference by passing messages in a cluster graph.