NUS National University of Singapore | School of Computing

CS5340
Uncertainty Modeling in AI

Lecture 6:
Factor Graph and the Junction Tree Algorithm

Asst. Prof. Lee Gim Hee

AY 2018/19

Semester 1

Course Schedule

Week	Date	Topic	Remarks
1	15 Aug	Introduction to probabilities and probability distributions	
2	22 Aug	Fitting probability models	Hari Raya Haji*
3	29 Aug	Bayesian networks (Directed graphical models)	
4	05 Sep	Markov random Fields (Undirected graphical models)	
5	12 Sep	I will be traveling	No Lecture
6	19 Sep	Variable elimination and belief propagation	
-	26 Sep	Recess week	No lecture
7	03 Oct	Factor graph and the junction tree algorithm	
8	10 Oct	Parameter learning with complete data	
9	17 Oct	Mixture models and the EM algorithm	
10	24 Oct	Hidden Markov Models (HMM)	
11	31 Oct	Monte Carlo inference (Sampling)	
12	07 Nov	Variational inference	
13	14 Nov	Graph-cut and alpha expansion	

* **Make-up lecture**: 25 Aug (Sat), 9.30am-12.30pm, LT 15

Acknowledgements

- A lot of slides and content of this lecture are adopted from:

1. Michael I. Jordan "An introduction to probabilistic graphical models", 2002. Chapters 4.2, 4.3 and 17
   http://people.eecs.berkeley.edu/~jordan/prelims/chapter4.pdf
   http://people.eecs.berkeley.edu/~jordan/prelims/chapter17.pdf

2. Daphne Koller and Nir Friedman, "Probabilistic graphical models“ Chapter 10

3. David Barber, "Bayesian reasoning and machine learning” Chapter 6

4. Kevin Murphy, "Machine learning: a probabilistic approach” Chapter 20.4

5. Christopher Bishop "Machine learning and pattern recognition“ Chapter 8.4.3

Learning Outcomes

- Students should be able to:

1. Represent a joint distribution with a factor graph, and use it to compute the marginal/conditional probabilities.

2. Use the max-product algorithm to find the maximal probability and its configurations.

3. Convert a DGM/UGM into the junction tree and use it to compute the marginal/conditional probabilities.

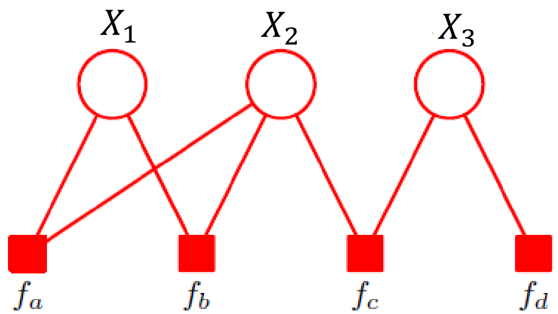
Factor Graphs

- DGMs and UGMs: allow a global function of several variables to be expressed as a product of factors over subsets of those variables.

- Factor graphs make this decomposition explicit by introducing additional nodes for the factors in addition to the nodes representing the variables.

- Unlike DGMs and UGMs, factor graphs are NOT designed for conditional independence, but for more explicit details of the factorization.

Factor Graphs: Graphical Representation



- A factor graph is a bipartite graph:

$$\mathcal{G}(\mathcal{V}, \mathcal{F}, \mathcal{E})$$

where

- vertices $\mathcal{V} \in \{X_1, \dots, X_n\}$: index the random variables,
- vertices $\mathcal{F} \in \{\dots, f_s, \dots\}$: index the factors and
- undirected edges $\mathcal{E}$: link each factor node f_s to all variable nodes X_s that f_s depends.

- We use round nodes to represent random variables and square nodes to represent factors.

Image source: “Pattern recognition and machine learning”, Christopher Bishop

Factor Graphs: Joint Distribution

- We write the joint distribution over a set of variables in the form of a product of factors:

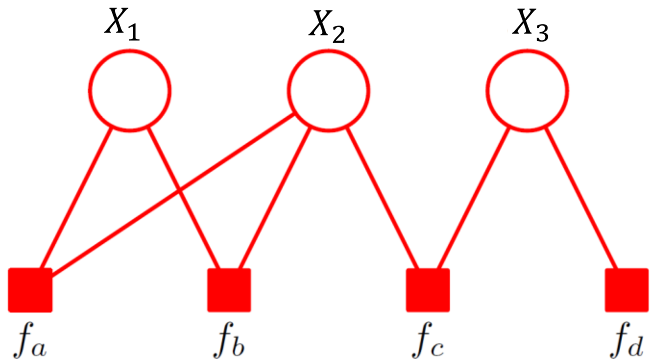
$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$

- Where X_s denotes a subset of the variables $\mathrm{X} \in \{X_1, \dots, X_n\}$.

- Each factor f_s is a function of a corresponding set of variables X_s.

Factor Graphs

Example:



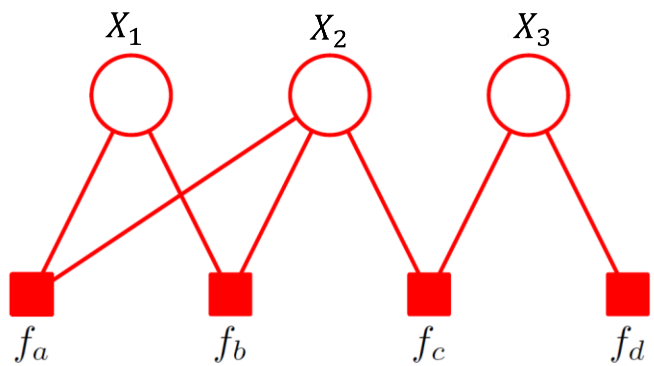
$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

- Note that there are two factors $f_a(x_1, x_2)$ and $f_b(x_1, x_2)$ that are defined over the same set of variables.
- In an undirected graph, product of two such factors would simply be lumped together into the same clique potential.

Image source: “Pattern recognition and machine learning”, Christopher Bishop

Factor Graphs

Example:



$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

- Similarly, $f_c(x_2, x_3)$ and $f_d(x_3)$ could be combined into a single potential over X_2 and X_3.
- The factor graph keeps such factors explicit, so is able to convey more detailed information about the underlying factorization.

Image source: "Pattern recognition and machine learning", Christopher Bishop

Convert DGM to Factor Graph

- Recall the factorization of DGMs is defined as:

$$p(x_1, \dots, x_N) = \prod_{i=1}^{N} p\left(x_i \middle| x_{\pi_i}\right)$$

- Convert a DGM into a factor graph by representing the local conditional distributions $p\left(x_i \middle| x_{\pi_i}\right)$ as factors $f_s(\mathrm{x}_s)$.

Convert UGM to Factor Graph

- Recall the factorization of UGMs is defined as:

$$p(\mathbf{y}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c|\boldsymbol{\theta}_c)$$

- Convert a UGM into a factor graph by representing the potential functions over the maximal cliques as factors $f_s(\mathrm{x}_s)$.

- Normalizing coefficient $1/Z$ can be viewed as a factor defined over the empty set of variables.

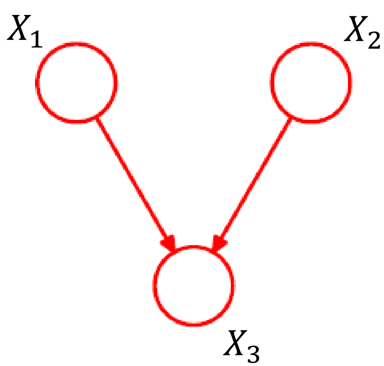
DGM/UGM to Factor Graph

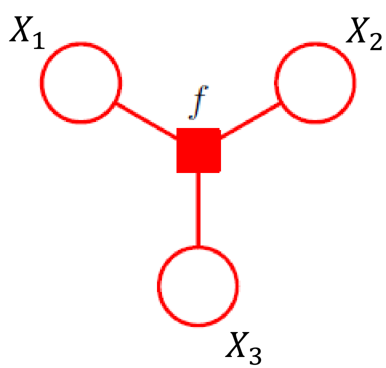
- Note that there may be several different factor graphs that correspond to the same DGM / UGM.

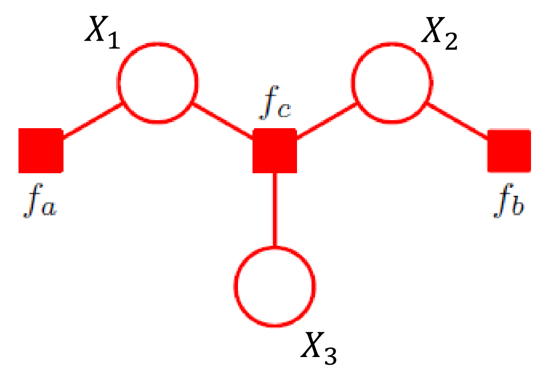
- Factor graphs to be more specific about the precise form of the factorization.

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

Example: Directed Graph







$p(x_1)p(x_2)p(x_3|x_1, x_2)$ $\quad f(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3|x_1, x_2)$

Two factor graphs representing the same distribution

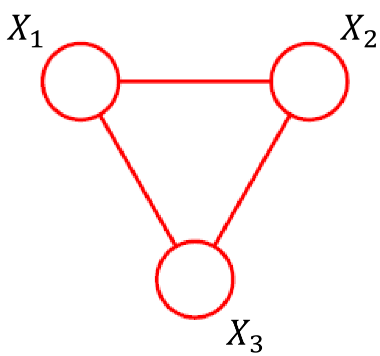
Image source: “Pattern recognition and machine learning”, Christopher Bishop

DGM/UGM to Factor Graph

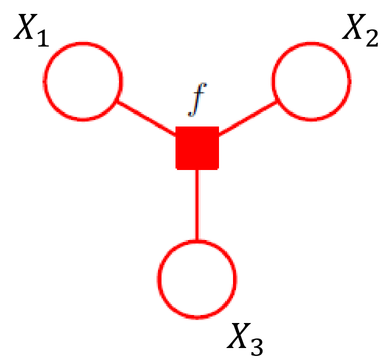
- Note that there may be several different factor graphs that correspond to the same DGM / UGM.
- Factor graphs to be more specific about the precise form of the factorization.

Example: Undirected Graph

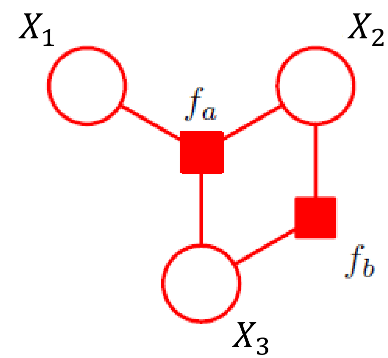
$$f_a(x_1, x_2, x_3) f_b(x_1, x_2) = \psi(x_1, x_2, x_3)$$



Single clique potential
$\psi(x_1, x_2, x_3)$



$$f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$$



Two factor graphs representing the same distribution

Image source: “Pattern recognition and machine learning”, Christopher Bishop

Factor Graphs: Sum-Product Algorithm

- Alternative representation for the sum-product algorithm for "tree-like" graphs.

- More importantly, some DGMs/UGMs with local cycles become a tree when converted to factor graphs.

Example: Turning local cycle into a tree

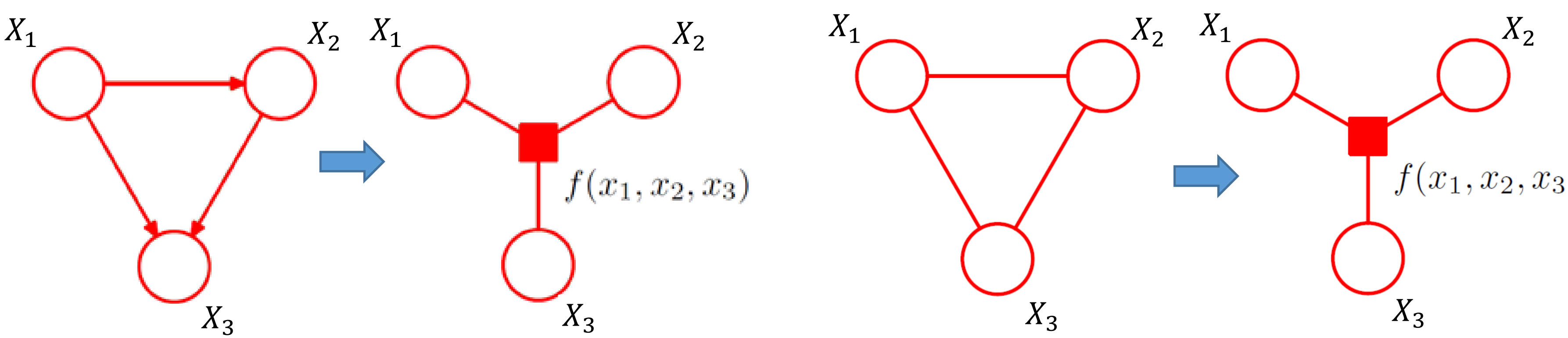
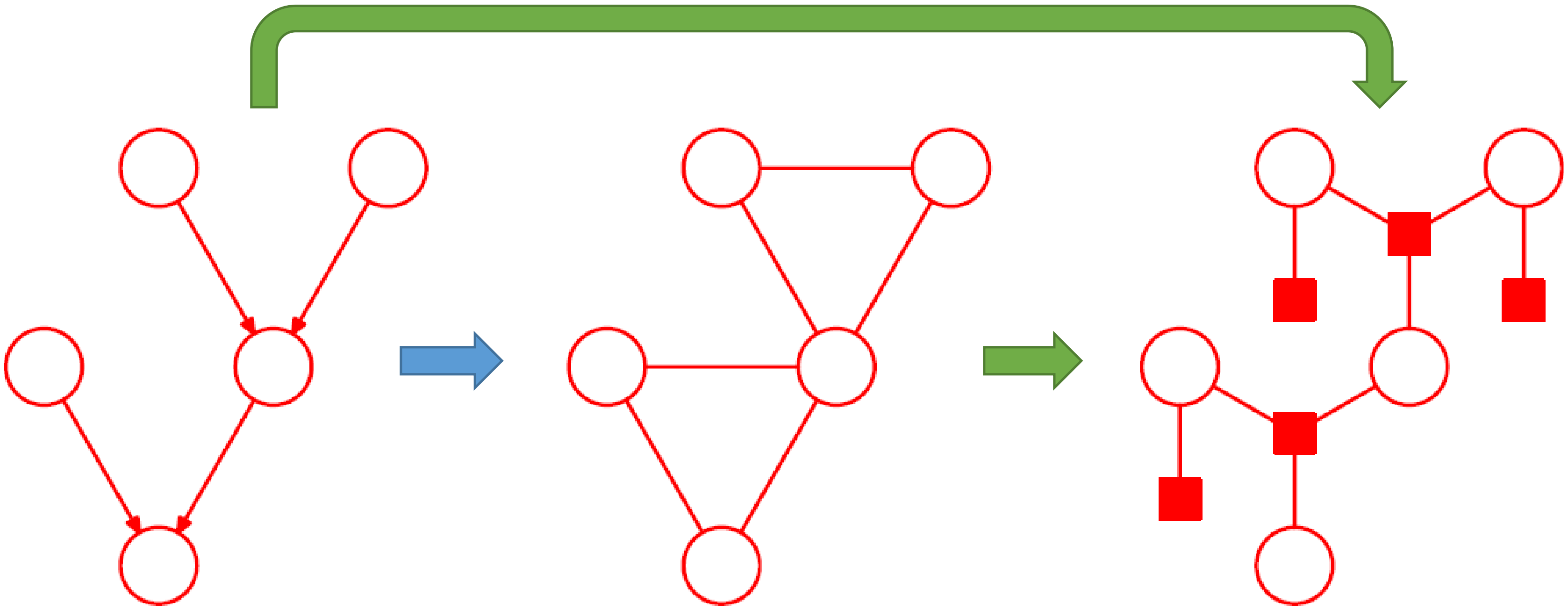


Image source: "Pattern recognition and machine learning", Christopher Bishop

Polytrees



- Cycles appear after directed to undirected graph conversion.
- Local cycles disappeared after factor graph conversion.
- Note the factor graph conversion can be directly from a DGM.

Image source: “Pattern recognition and machine learning”, Christopher Bishop

Factor Graphs: Sum-Product Algorithm

- **Our goal**: Compute all singleton marginal probabilities under the factorized representation of the joint probability.

- As in the earlier Sum-Product algorithm, we define two kinds of messages:

  1. Messages υ: flow from variable to factor nodes.
  2. Messages μ: flow from factor to variable nodes.

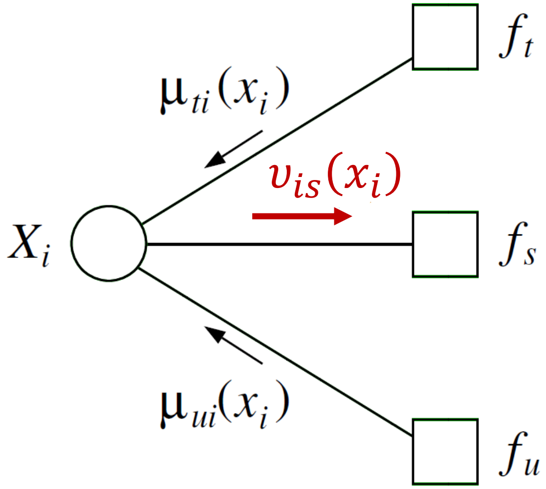
Neighborhood Sets of a Node

- $N(s) \subset \mathcal{V}$: Set of neighbors of a factor node $s \in \mathcal{F}$.
- $N(s)$ refers to the indices of all variables referenced by the factor f_s.
- $N(i) \subset \mathcal{F}$: Set of neighbors of a variable node $i \in \mathcal{V}$.
- $N(i)$ for a variable node X_i refers to the set of all factors that referenced X_i.

Messages from Variable to Factor Nodes

- Message $\upsilon_{is}(x_i)$ flows from the variable node X_i to the factor node f_s:

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



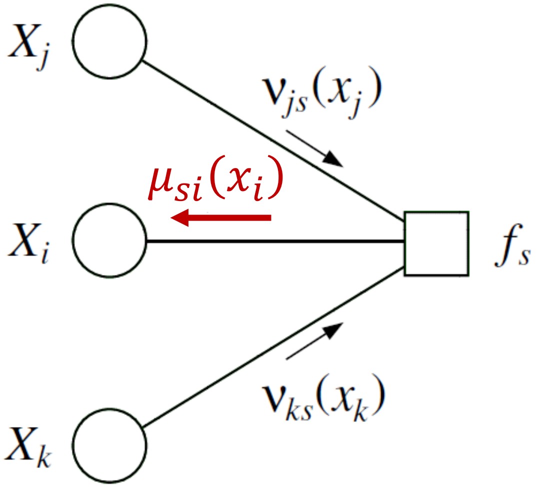
- The product is taken over all incoming messages to the variable node X_i, other than the factor node f_s.

Image source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Messages from Factor to Variable Nodes

- Message $\mu_{si}(x_i)$ flows from the factor node f_s to the variable node X_i :

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



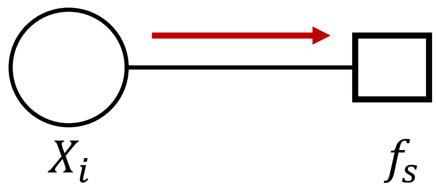
- The product is taken over all incoming messages to the factor node f_s, other than the variable node X_i.

Image source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Messages From The Leaf Nodes

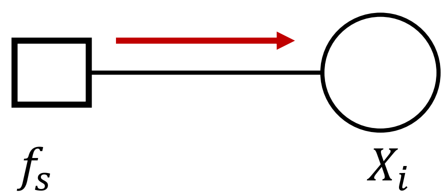
- Message from a leaf variable node to factor node:

$$v_{is}(x_i) = 1$$



- Message from a leaf factor node to variable node:

$$\mu_{si}(x_i) = f_s(x_i)$$



Message-Passing Protocol

A node can send a message to a neighboring node when (and only when) it has received messages from all of its other neighbors.

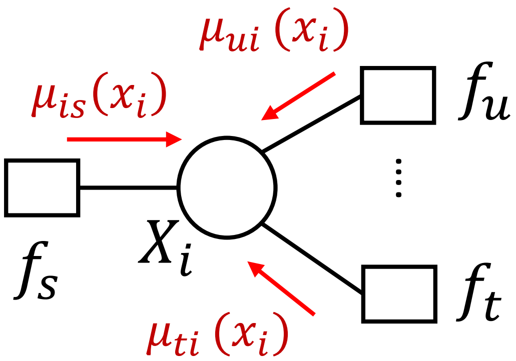
Applies to both variable and factor nodes.

Marginal Probability of a Node

- Once a node X_i has received the messages from all its neighbors, the marginal probability is given by:

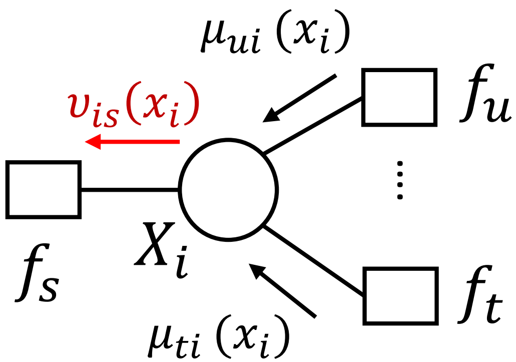
$$p(x_i) \propto \prod_{s \in \mathcal{N}(i)} \mu_{si}(x_i)$$

$$= \nu_{is}(x_i)\mu_{si}(x_i)$$



since

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



Factor Tree Sum-Product Algorithm

$\textsc{Sum-Product}(\mathcal{T}, E)$ // main steps of **Sum-Product algorithm**

1. $\quad\textsc{Evidence}(E)$
   $\quad f = \textsc{ChooseRoot}(\mathcal{V})$
2. $\quad$**for** $s \in \mathcal{N}(f)$
   $\quad\quad \mu\text{-}\textsc{Collect}(f, s)$
3. $\quad$**for** $s \in \mathcal{N}(f)$
   $\quad\quad \nu\text{-}\textsc{Distribute}(f, s)$
4. $\quad$**for** $i \in \mathcal{V}$
   $\quad\quad \textsc{ComputeMarginal}(i)$

1. $\textsc{Evidence}(E)$ // add **evidence potentials** (convert conditioning into marginalization)
   for $i \in E$
   $\quad \psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$
   for $i \notin E$
   $\quad \psi^E(x_i) = \psi(x_i)$

2. $\mu\text{-}\textsc{Collect}(i, s)$ // recursively collect messages from leaves to root
   $\boxed{\textbf{for } j \in \mathcal{N}(s)\backslash i \quad \nu\text{-}\textsc{Collect}(s, j)}$
   $\mu\text{-}\textsc{SendMessage}(s, i)$

$$\prod_{j \in \mathcal{N}(s)\backslash i} \nu_{js}(x_j)$$

$\boxed{\nu\text{-}\textsc{Collect}(s, i)}$
for $t \in \mathcal{N}(i)\backslash s$
$\quad \mu\text{-}\textsc{Collect}(i, t)$
$\nu\text{-}\textsc{SendMessage}(i, s)$

Message from variable node X_i to the factor node f_s:
$\nu\text{-}\textsc{SendMessage}(i, s)$

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

Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Factor Tree Sum-Product Algorithm

$\textsc{Sum-Product}(\mathcal{T}, E)$ // main steps of **Sum-Product algorithm**

1. $\textsc{Evidence}(E)$
   $f = \textsc{ChooseRoot}(\mathcal{V})$
2. **for** $s \in \mathcal{N}(f)$
   $\mu\text{-}\textsc{Collect}(f, s)$
3. **for** $s \in \mathcal{N}(f)$
   $\nu\text{-}\textsc{Distribute}(f, s)$
4. **for** $i \in \mathcal{V}$
   $\textsc{ComputeMarginal}(i)$

1. $\textsc{Evidence}(E)$ // add **evidence potentials** (convert conditioning into marginalization)
   for $i \in E$
   $\psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$
   for $i \notin E$
   $\psi^E(x_i) = \psi(x_i)$

2. $\mu\text{-}\textsc{Collect}(i, s)$ // recursively collect messages from leaves to root
   for $j \in \mathcal{N}(s)\backslash i$
   $\nu\text{-}\textsc{Collect}(s, j)$
   $\boxed{\mu\text{-}\textsc{SendMessage}(s, i)}$

   $\nu\text{-}\textsc{Collect}(s, i)$
   for $t \in \mathcal{N}(i)\backslash s$
   $\mu\text{-}\textsc{Collect}(i, t)$
   $\nu\text{-}\textsc{SendMessage}(i, s)$

Message from factor node f_s to the variable node X_i:
$\mu\text{-}\textsc{SendMessage}(s, i)$

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

Message from variable node X_i to the factor node f_s:
$\nu\text{-}\textsc{SendMessage}(i, s)$

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

Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Factor Tree Sum-Product Algorithm

// main steps of **Sum-Product algorithm**

$\textsc{Sum-Product}(\mathcal{T}, E)$

1. $\textsc{Evidence}(E)$

   $f = \textsc{ChooseRoot}(\mathcal{V})$

2. **for** $s \in \mathcal{N}(f)$

   $\mu\text{-}\textsc{Collect}(f, s)$

3. **for** $s \in \mathcal{N}(f)$

   $\nu\text{-}\textsc{Distribute}(f, s)$

4. **for** $i \in \mathcal{V}$

   $\textsc{ComputeMarginal}(i)$

3. $\nu\text{-}\textsc{Distribute}(i, s)$

   $\boxed{\nu\text{-}\textsc{SendMessage}(i, s)}$

   for $j \in \mathcal{N}(s) \backslash i$

   $\mu\text{-}\textsc{Distribute}(s, j)$

$\mu\text{-}\textsc{Distribute}(s, i)$

$\mu\text{-}\textsc{SendMessage}(s, i)$

for $t \in \mathcal{N}(i) \backslash s$

$\nu\text{-}\textsc{Distribute}(i, t)$

// distribute messages from root to leaves

Message from variable node X_i to the factor node f_s:

$\nu\text{-}\textsc{SendMessage}(i, s)$

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

4. $\textsc{ComputeMarginal}(i)$

   $p(x_i) \propto \nu_{is}(x_i) \mu_{si}(x_i)$

// compute **marginal probability**

Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Factor Tree Sum-Product Algorithm

$\textsc{Sum-Product}(\mathcal{T}, E)$ — // main steps of **Sum-Product algorithm**

1. $\textsc{Evidence}(E)$
   $f = \textsc{ChooseRoot}(\mathcal{V})$
2. **for** $s \in \mathcal{N}(f)$
   $\mu\text{-}\textsc{Collect}(f, s)$
3. **for** $s \in \mathcal{N}(f)$
   $\nu\text{-}\textsc{Distribute}(f, s)$
4. **for** $i \in \mathcal{V}$
   $\textsc{ComputeMarginal}(i)$

3. $\nu\text{-}\textsc{Distribute}(i, s)$ — // distribute messages from root to leaves
   $\nu\text{-}\textsc{SendMessage}(i, s)$
   for $j \in \mathcal{N}(s) \backslash i$
   $\mu\text{-}\textsc{Distribute}(s, j)$

   $\mu\text{-}\textsc{Distribute}(s, i)$
   $\mu\text{-}\textsc{SendMessage}(s, i)$
   for $t \in \mathcal{N}(i) \backslash s$
   $\nu\text{-}\textsc{Distribute}(i, t)$

Message from variable node X_i to the factor node f_s:

$\nu\text{-}\textsc{SendMessage}(i, s)$

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

Message from factor node f_s to the variable node X_i:

$\mu\text{-}\textsc{SendMessage}(s, i)$

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

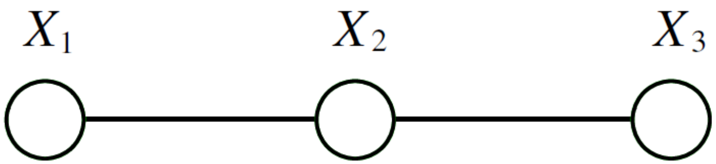
4. $\textsc{ComputeMarginal}(i)$ — // compute **marginal probability**
   $p(x_i) \propto \nu_{is}(x_i) \mu_{si}(x_i)$

Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

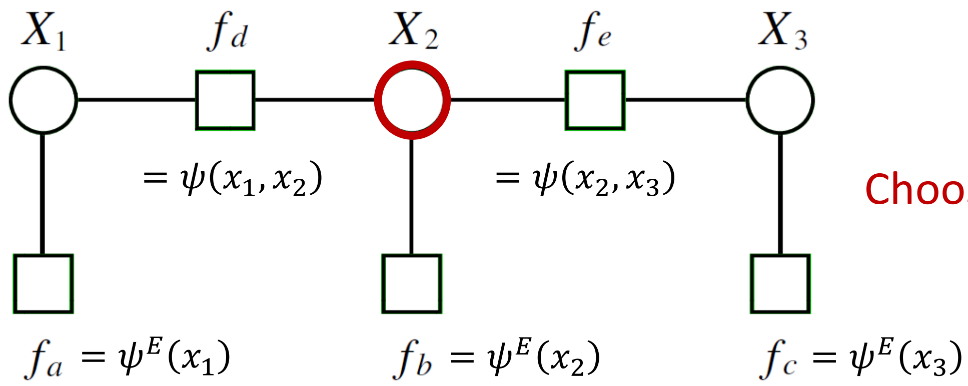
Factor Tree Sum-Product Algorithm

Example:

$$p(x|\bar{x}_E) = \frac{1}{Z^E}\left(\psi^E(x_1)\psi^E(x_2)\psi^E(x_3)\psi(x_1,x_2)\psi(x_2,x_3)\right)$$



Convert UGM into a factor graph

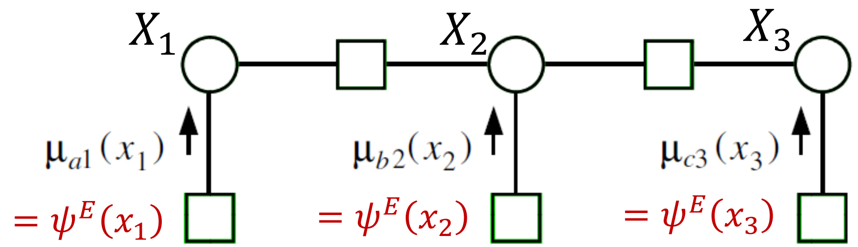


Choose X_2 as root node

Image Source: “An introduction to probabilistic graphical models”, Michael I. Jordan, 2002.

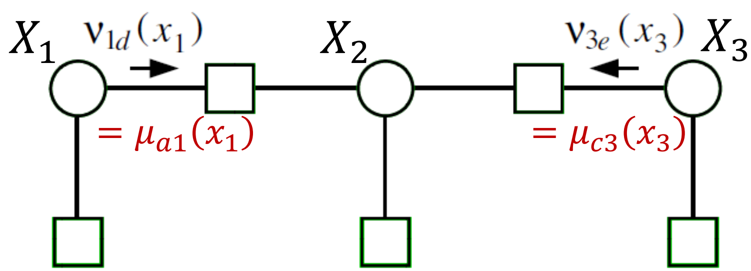
Factor Tree Sum-Product Algorithm

Example:



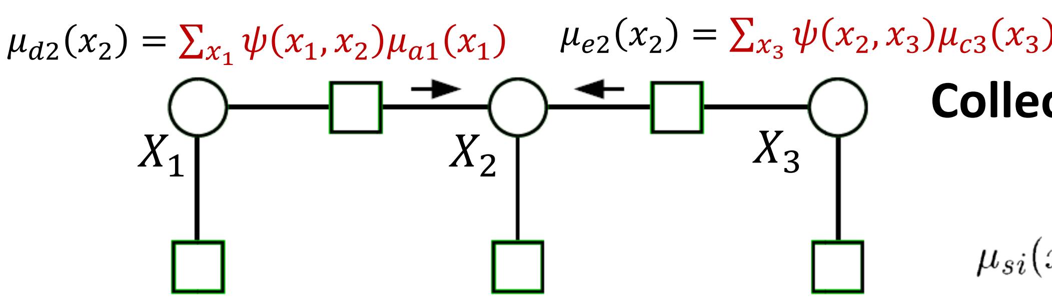
Collect messages from leaf nodes:

$$\mu_{si}(x_i) = f_s(x_i) = \psi^E(x_i)$$



Collect variable to factor messages:

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



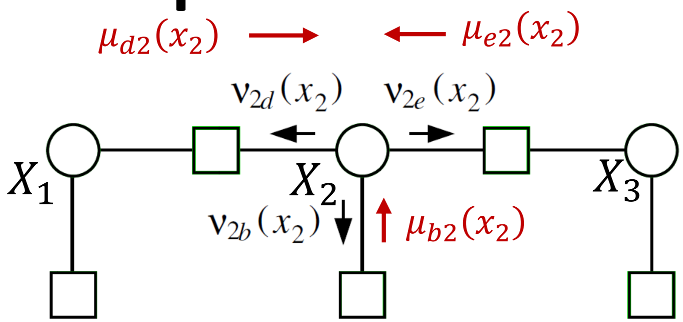
Collect factor to variable messages:

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

Image Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Factor Tree Sum-Product Algorithm

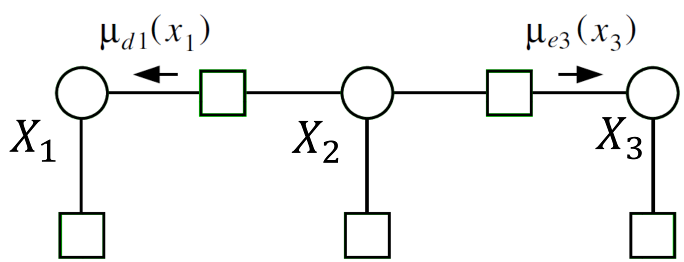
Example:



Distribute variable to factor messages:

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

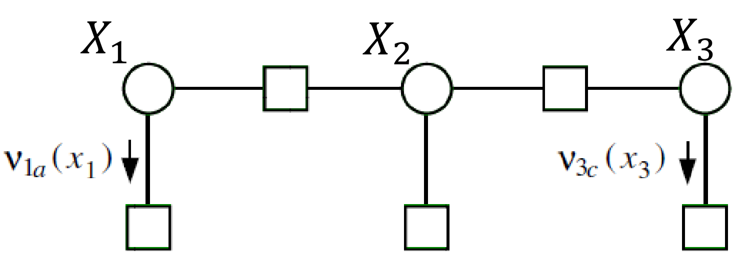
$$\upsilon_{2b}(x_2) = \mu_{d2}(x_2)\mu_{e2}(x_2)$$
$$\upsilon_{2d}(x_2) = \mu_{b2}(x_2)\mu_{e2}(x_2)$$
$$\upsilon_{2e}(x_2) = \mu_{b2}(x_2)\mu_{d2}(x_2)$$



Distribute factor to variable messages:

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

$$\mu_{d1}(x_1) = \sum_{x_2} \psi(x_1, x_2)\upsilon_{2d}(x_2)$$
$$\mu_{e3}(x_3) = \sum_{x_2} \psi(x_2, x_3)\upsilon_{2e}(x_2)$$



Distribute variable to factor messages:

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

$$\upsilon_{1a}(x_1) = \mu_{d1}(x_1), \quad \upsilon_{3c}(x_3) = \mu_{e3}(x_3)$$

Image Source: “An introduction to probabilistic graphical models”, Michael I. Jordan, 2002.

Relation Between Sum-Product for UGMs and Factor Graph

- $m_{ji}(x_i)$ in the undirected graph is equal to $\mu_{si}(x_i)$ in the factor graph!

Proof:

UGM:

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

Factor Graph:

$$\begin{aligned}
\mu_{si}(x_i) &= \sum_{x_{\mathcal{N}(s) \backslash i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \backslash i} \nu_{js}(x_j) \right) \\
&= \sum_{x_j} \psi(x_i, x_j) \nu_{js}(x_j) \\
&= \sum_{x_j} \psi(x_i, x_j) \prod_{t \in \mathcal{N}(j) \backslash s} \mu_{tj}(x_j) \\
&= \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{t \in \mathcal{N}'(j) \backslash s} \mu_{tj}(x_j) \right)
\end{aligned}$$

$N'(j)$ denotes the neighbourhood of X_j, omitting the singleton factor node associated with $\psi^E(x_j)$.

Maximum a Posterior Probabilities

- Marginalization problem: summing over all configurations of sets of random variables.
- Maximum a Posterior (MAP) problem: maximizing over all sets of random variables.
- Two aspects to MAP:
    1. Finding the maximal probability.
    2. Finding a configuration that achieves the maximal probability.

Maximal Probability

- Given a probability distribution $p(x \mid \bar{x}_E)$, the maximum a posterior probability is given by:

$$\max_x p(x \mid \bar{x}_E) = \max_x \frac{p(x, \bar{x}_E)}{p(\bar{x}_E)}$$

Can be removed since we are finding max over X.

$$= \max_x p(x, \bar{x}_E)$$

$$= \max_x p(x)\delta(x_E, \bar{x}_E)$$

$$= \max_x p(x)^E$$

where

- $\bar{X}_E$ is the set of observed variables, and
- $p(x)^E$ is the unnormalized representation of the conditional probability $p(x, \bar{x}_E)$.

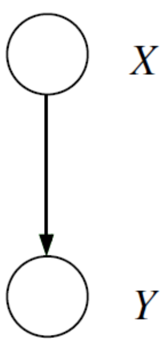
Fallacy

- Can we solve the MAP problem by computing the:

  1. marginal probability for each variable, and
  2. assignment of each variable that maximizes its individual marginal?

NO!!!

Illustration:



x	
1	.6
2	.2
3	.2

$p(x)$

x \ y	1	2	3
1	0	.6	.4
2	1	0	0
3	1	0	0

$p(y \mid x)$

1	2	3
.4	.36	.24

$p(y)$

Marginal probabilities:

$$\max_x p(x) = p(x = 1) = 0.6$$

$$\max_y p(y) = p(y = 1) = 0.4$$

But

$$\max_{x,y} p(x, y) = p(x = 1, y = 2) = 0.36$$

Image Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

From Marginal to MAP Algorithms

- Distributive law of multiplication over addition:

$$a.b_1 + a.b_2 + \ \dots + a.b_n = a.(b_1 + b_2 + \cdots + b_n)$$

- Plays a key role in elimination and sum-product algorithms:

$$p(x_1, x_2, \dots, x_5) = \sum_{x_6} \underbrace{p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)}_{a} p(x_6|x_2, x_5)$$

$$= \sum_{x_6} a.p(x_6|x_2, x_5)$$

$$= a.p(x_6 = 0|x_2, x_5) + \cdots + a.p(x_6 = k|x_2, x_5)$$

$$= a.\big(p(x_6 = 0|x_2, x_5) + \cdots + p(x_6 = k|x_2, x_5)\big)$$ (Distributive law)

$$= p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3) \sum_{x_6} p(x_6|x_2, x_5)$$

From Marginal to MAP Algorithms

- Distributive law applies to the "max" operator too!

$$\max(a.b_1, a.b_2, \ldots, a.b_n) = a.\max(b_1 + b_2 + \cdots + b_n)$$

- Turn the elimination algorithm into the "MAP-elimination" algorithm by replacing the "sum" with "max" operator:

$$\max_{x_6} p(x_1, x_2, \ldots, x_6) = \underbrace{\max_{x_6}}_{} p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2, x_5)$$

"max" operator can be pushed in!

$$= \underbrace{p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)}_{\text{independent of } x_6}\max_{x_6} p(x_6|x_2, x_5)$$

- Becomes the "max-product" algorithm.

MAP-Elimination Algorithm

MAP-ELIMINATE($\mathcal{G}, E$) // main steps of the "MAP-Elimination Algorithm"

1. INITIALIZE($\mathcal{G}$)
2. EVIDENCE(E)
3. UPDATE($\mathcal{G}$)
4. MAXIMUM

1. INITIALIZE($\mathcal{G}$) // choose elimination ordering, and add local condition probabilities in **active list**

    choose an ordering I // **same** as the "variable elimination algorithm"

    for each node X_i in $\mathcal{V}$

        place $p(x_i \mid x_{\pi_i})$ on the active list

2. EVIDENCE(E) // add evidence potentials in **active list**

    for each i in E // **same** as the "variable elimination algorithm"

        place $\delta(x_i, \bar{x}_i)$ on the active list

3. UPDATE($\mathcal{G}$) // **maximization**, and update active list

    for each i in I

        find all potentials from the active list that reference x_i and remove them from the active list

        let $\phi_i^{\max}(x_{T_i})$ denote the product of these potentials

        let $\boxed{m_i^{\max}(x_{S_i}) = \max_{x_i} \phi_i^{\max}(x_{T_i})}$

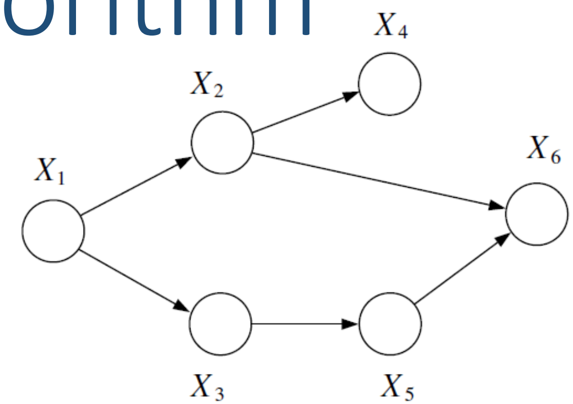
        place $m_i^{\max}(x_{S_i})$ on the active list

4. MAXIMUM

    $\max_x p^E(x) =$ the scalar value on the active list

Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

MAP-Elimination Algorithm



Example:

Elimination order: $I = \{6, 5, 4, 3, 2, 1\}$

$$p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2, x_5)$$

$$\max_{x} p(x_1, x_2, x_3, x_5|\bar{x}_6) = \max_{x_1} \max_{x_2} \max_{x_3} \max_{x_4} \max_{x_5} \max_{x_6} \frac{p(x_1, x_2, x_3, x_5, \bar{x}_6)}{p(\bar{x}_6)}$$

$$= \max_{x_1} \max_{x_2} \max_{x_3} \max_{x_4} \max_{x_5} \max_{x_6} p(x_1, x_2, x_3, x_5, \bar{x}_6)$$

$$= \max_{x_1} \max_{x_2} \max_{x_3} \max_{x_4} \max_{x_5} \max_{x_6} p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2, x_5)\delta(x_6, \bar{x}_6)$$

$$= \underbrace{\max_{x_1} p(x_1) \underbrace{\max_{x_2} p(x_2|x_1) \underbrace{\max_{x_3} p(x_3|x_1) \underbrace{\max_{x_4} p(x_4|x_2) \underbrace{\max_{x_5} p(x_5|x_3) \underbrace{\max_{x_6} p(x_6|x_2, x_5)\delta(x_6, \bar{x}_6)}_{m_6(x_2, x_5)}}_{m_5(x_2, x_3)}}_{m_4(x_2, x_3)}}_{m_3(x_1, x_2)}}_{m_2(x_1)}}_{m_1(x_1)}$$

Image Source: “An introduction to probabilistic graphical models”, Michael I. Jordan, 2002.

Underflow Problem

- Products of probabilities (numbers between 0 and 1) tend to underflow!

- Can be overcome by transforming to the monotone log scale:

$$\max_x p^E(x) = \max_x \log p^E(x)$$

- Fortunately, the distributive law still holds:

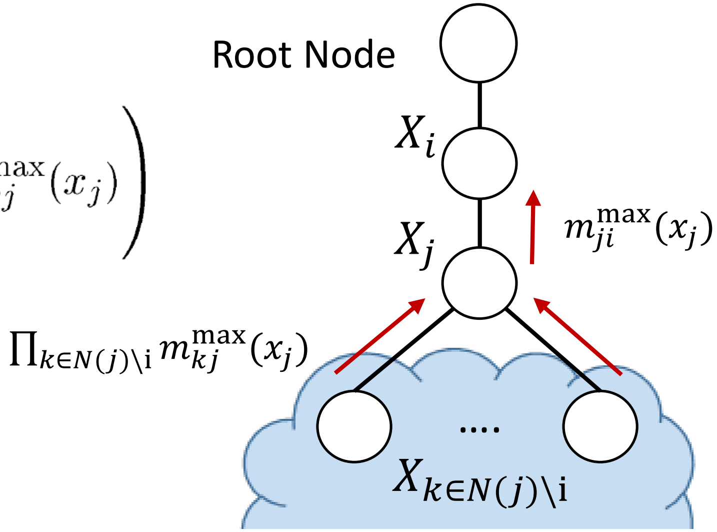
$$\max(a + b_1, a + b_2, \dots, a + b_n) = a + \max(b_1, b_2, \dots, b_n)$$

- Turns the "max-product" algorithm into the "max-sum" algorithm.

Max-Product Algorithm for Trees

- Find the MAP probability for a tree.
- We choose any node X_f as the root of the tree, and messages are propagated (inward pass) from the leaves to the root.
- Message from X_j to X_i (closer to root):

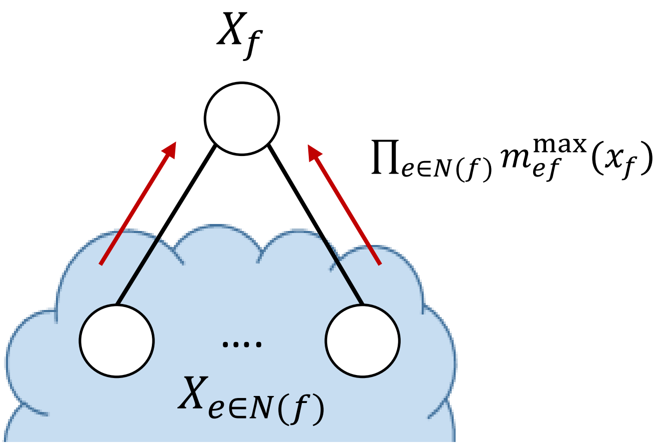
$$m_{ji}^{\max}(x_i) = \max_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \backslash i} m_{kj}^{\max}(x_j) \right)$$



Max-Product Algorithm for Trees

- Collect all messages at the root and compute the MAP probability as:

$$\max_{x} p^E(x) = \max_{x_i} \left(\psi^E(x_f) \prod_{e \in N(f)} m_{ef}^{\max}(x_f) \right)$$



- Do we need to pass the messages back to the leaves?

No!

MAP probabilities for all choices of the root node are the same.

Maximum a Posteriori Configurations

- This is the problem of finding a configuration x^* such that:

$$x^* \in \underset{x}{\operatorname{argmax}}\, p^E(x)$$

- Making use of the messages to the root X_f from the sum-product algorithm, we obtain a value:

$$x_f^* \in \arg\max_{x_f} \left(\psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}^{\max}(x_f) \right)$$

that necessarily belongs to a maximum configuration.

Maximum a Posteriori Configurations

- Can we perform an outward pass of the messages from the root to leaves so that we can find the MAP configurations for all x?

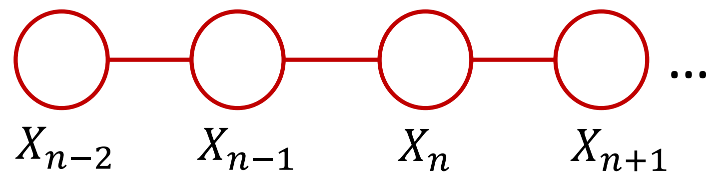
NO!

- No guarantee that the values x^* found this way belong to the same maximizing configuration.

Maximum a Posteriori Configurations

Example:

A lattice, or trellis, diagram shows two sets of configurations (black paths) in a chain model that give rise to the same MAP probability.



$x_n \in \{1,2,3\}$

Trellis diagram shows each possible state of the random variable.

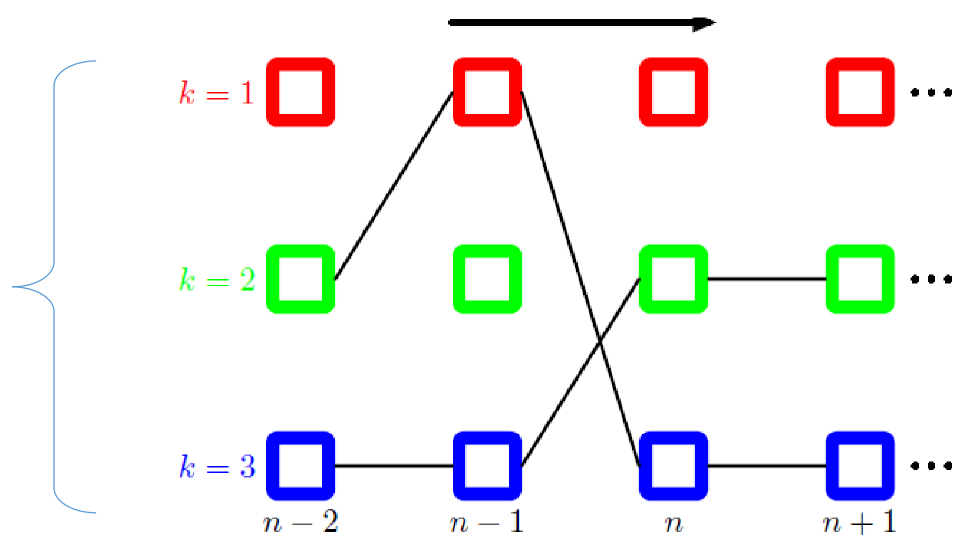


Image source: "Pattern recognition and machine learning", Christopher Bishop

Max-Product Algorithm for Trees

- **Solution**: we also have to record the maximizing values in a table $\delta_{ji}(x_i)$ when a message $m_{ji}^{\max}(x_i)$ is sent from X_j to X_i (closer to root):

$$\delta_{ji}(x_i) \in \arg\max_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in \mathcal{N}(j)\setminus i} m_{kj}^{\max}(x_j) \right)$$

- More precisely, for each X_i, the function $\delta_{ji}(x_i)$ picks out a value of X_j (can be several) that achieves the maximum.

Max-Product Algorithm for Trees

- Having defined $\delta_{ji}(x_i)$ during the inward pass, we use $\delta_{ji}(x_i)$ to define a consistent maximizing configuration during an outward pass:

1. Choose a maximizing value x_f^* at the root X_f.
2. Set $x_e^* = \delta_{ef}(x_f^*)$ for each $e \in N(f)$.
3. Procedure continues outward to the leaves.

Max-Product Algorithm for Trees

$\textsc{Max-Product}(\mathcal{T},\ E)$ // main steps of the "MAP-Product Algorithm" for a tree $\mathcal{T}(\mathcal{V},\mathcal{E})$

$\quad \textsc{Evidence}(E)$

$\quad f = \textsc{ChooseRoot}(\mathcal{V})$

1. $\quad$ **for** $e \in \mathcal{N}(f)$

$\quad\quad \textsc{Collect}(f, e)$

$\quad \boxed{MAP = \max_{x_f}(\psi^E(x_f)\prod_{e\in\mathcal{N}(f)} m_{ef}^{\max}(x_f))}$ // compute MAP probability at root

$\quad \boxed{x_f^* = \arg\max_{x_f}(\psi^E(x_f)\prod_{e\in\mathcal{N}(f)} m_{ef}^{\max}(x_f))}$ // get MAP configuration at root

2. $\quad$ **for** $e \in \mathcal{N}(f)$

$\quad\quad \textsc{Distribute}(f, e)$

1. $\textsc{Collect}(i, j)$ // inward message passing

$\quad$ **for** $k \in \mathcal{N}(j)\backslash i$

$\quad\quad \textsc{Collect}(j, k)$

$\quad \underline{\textsc{SendMessage}(j, i)}$

2. $\textsc{Distribute}(i, j)$ // outward message passing

$\quad \underline{\textsc{SetValue}(i, j)}$

$\quad$ **for** $k \in \mathcal{N}(j)\backslash i$

$\quad\quad \textsc{Distribute}(j, k)$

$\underline{\textsc{SendMessage}(j, i)}$

$$m_{ji}^{\max}(x_i) = \max_{x_j}(\psi^E(x_j)\psi(x_i, x_j)\prod_{k\in\mathcal{N}(j)\backslash i} m_{kj}^{\max}(x_j))$$ // compute MAP probability message

$$\delta_{ji}(x_i) \in \arg\max_{x_j}(\psi^E(x_j)\psi(x_i, x_j)\prod_{k\in\mathcal{N}(j)\backslash i} m_{kj}^{\max}(x_j))$$ // get MAP configurations

$\underline{\textsc{SetValue}(i, j)}$ // get MAP configuration in outward pass

$\quad x_j^* = \delta_{ji}(x_i^*)$

From Variable Elimination to Junction Tree

- Variable Elimination is query sensitive: we must re-run the entire algorithm for each query node.

- The Junction Tree algorithm generalizes Variable Elimination to avoid this.

From Variable Elimination to Junction Tree

- Main idea behind Junction Trees:
  - Probability distributions corresponding to loopy undirected graphs can be re-parameterized as trees.
  - We can run the Sum-Product algorithm on the tree re-parameterization.

Cluster Graphs

- Undirected graph such that:

  1. Nodes are clusters $C_i \subseteq \{X_1, \dots, X_n\}$, where X_i are the random variables.
  2. Edge between C_i and C_j associated with sepset $S_{ij} = C_i \cap C_j$.

- Family preservation: given a set of potentials $\Psi \in \{\psi_1, \dots, \psi_k\}$ from an UGM, we assign each ψ_k to a cluster $C_{\alpha(k)}$ s.t. $\text{Scope}[\psi_k] \subseteq C_{\alpha(k)}$.

- Cluster potential is defined as:

$$\phi_i(C_i) = \prod_{k:\alpha(k)=i} \psi_k$$

Cluster Graphs

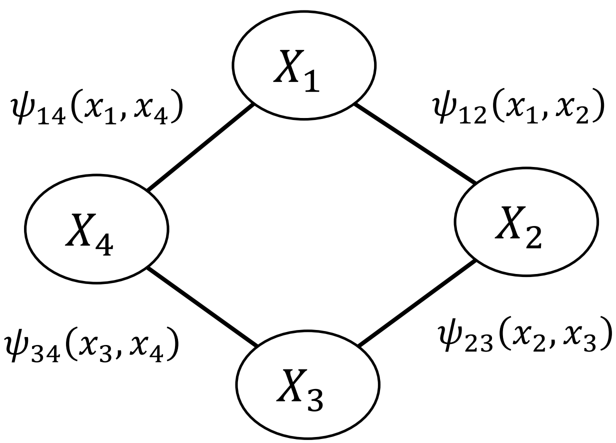
Example:

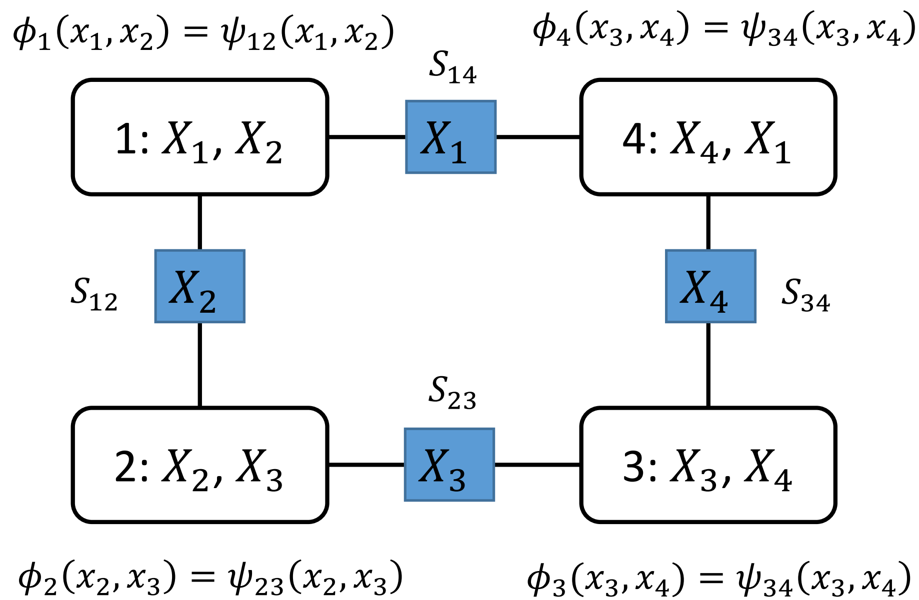
Cluster Graph

Sepset: $S_{ij} \subseteq C_i \cap C_j$

Cluster potential: $\phi_i(C_i) = \prod_{k:\alpha(k)=i} \psi_k$

Undirected Graphical Model





Adapted from: "Probabilitistic Graphical Models", Daphne Koller

Running Intersection Property: Junction Tree Property:

- For each pair of clusters C_i, C_j and variable $X \in C_i \cap C_j$:

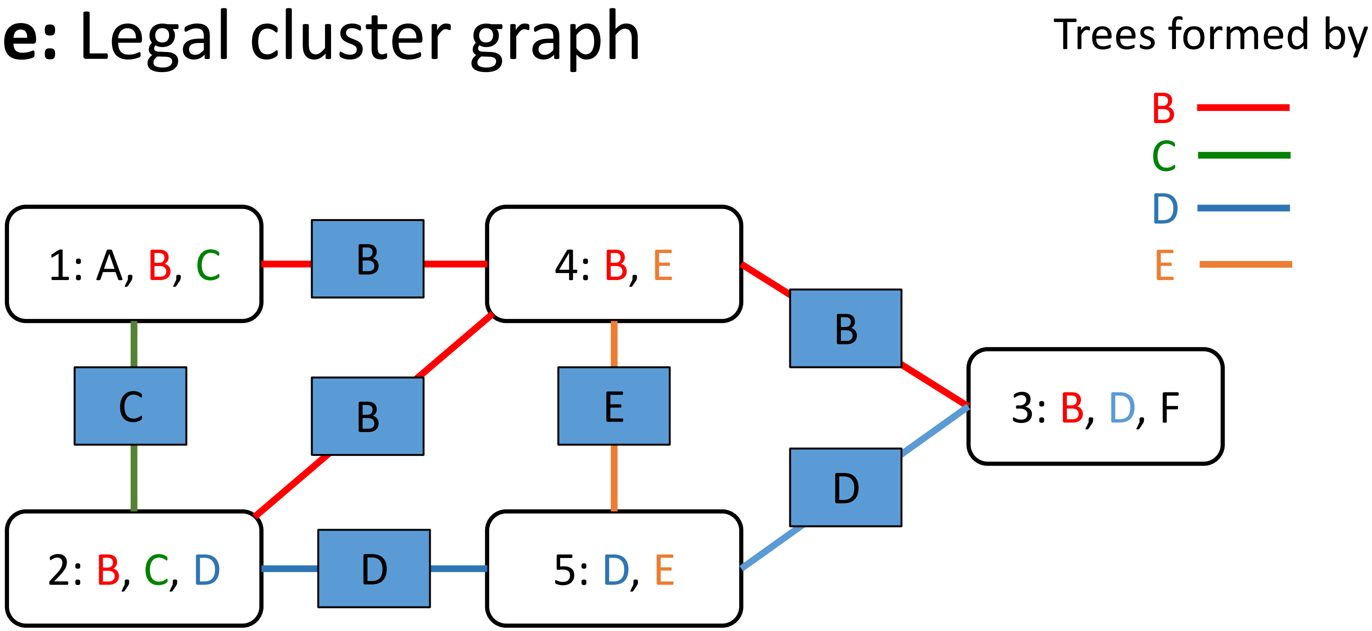
  There exists an unique path between C_i and C_j for which all clusters and sepsets contain X.

- Equivalently: For any X, the set of clusters and sepsets containing X form a tree.

Running Intersection Property: Junction Tree Property:

- A valid cluster graph must fulfil the running intersection property.

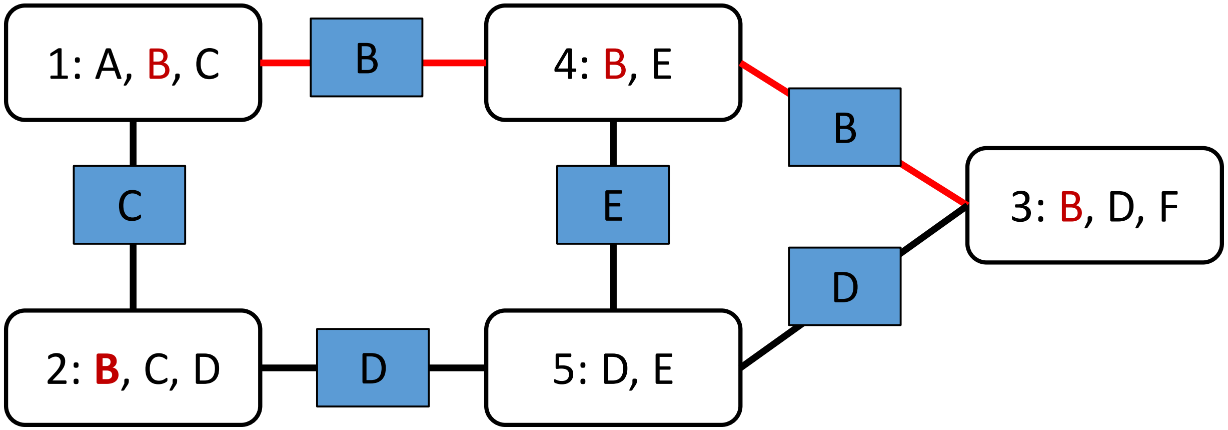
Example: Legal cluster graph



Adapted from: "Probabilitistic Graphical Models", Daphne Koller

Running Intersection Property: Junction Tree Property:

Example: Illegal cluster graph $\mathcal{I}$

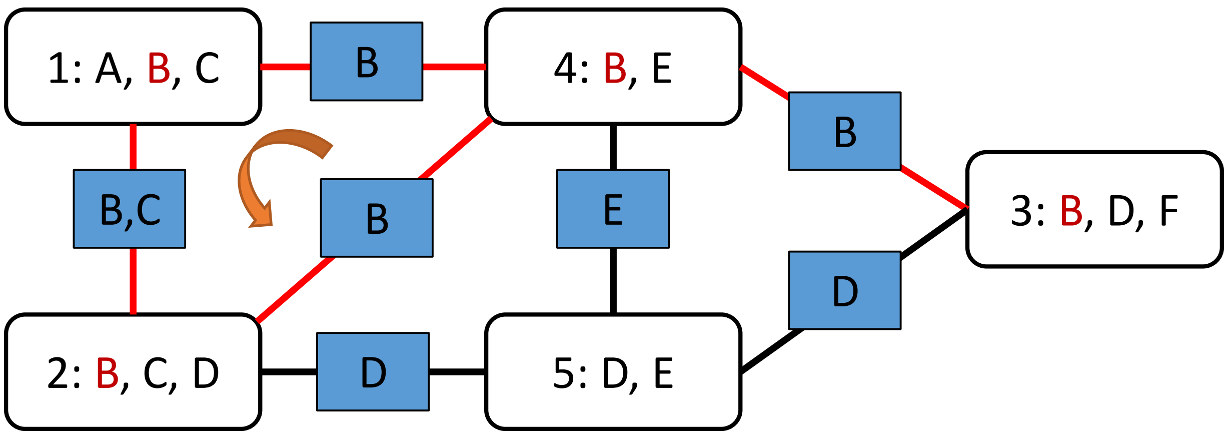


B is disconnected from the path!

Adapted from: "Probabilitistic Graphical Models", Daphne Koller

Running Intersection Property: Junction Tree Property:

Example: Illegal cluster graph II

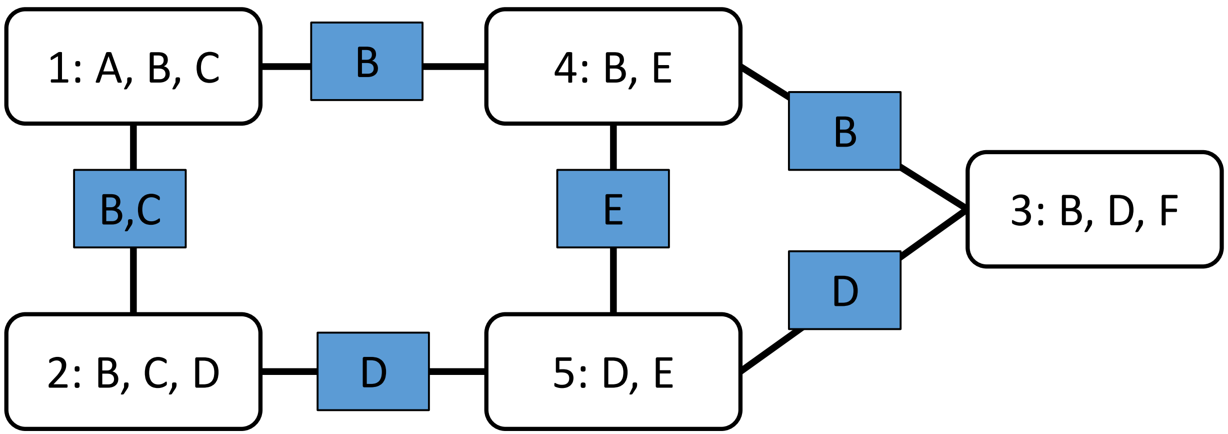


B forms a cycle!

Adapted from: "Probabilitistic Graphical Models", Daphne Koller

Running Intersection Property: Junction Tree Property:

Example: Alternative legal cluster graph



Adapted from: "Probabilitistic Graphical Models", Daphne Koller

Clique Trees a.k.a. Junction Trees

- A cluster graph without cycles is known as the cluster tree.
- A cluster tree that fulfills the running intersection property is called the clique tree, a.k.a. junction tree.
- We refer to a "cluster" in a clique tree as "clique", and "cluster potential" as "clique potential".

Clique Trees a.k.a. Junction Trees

We will first look at how to compute all marginals via the junction tree, before looking at how to convert a DGM/UGM into a junction tree.

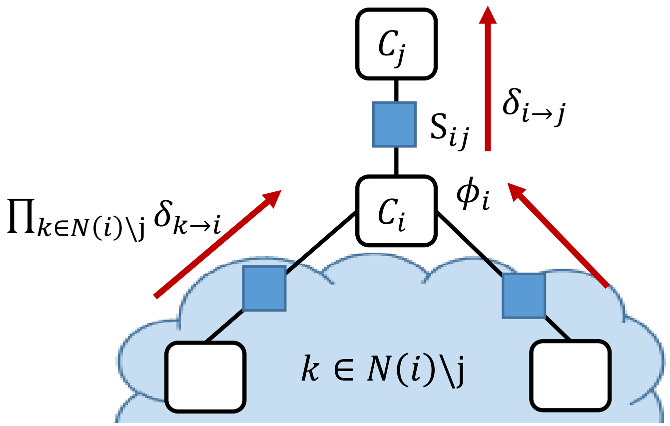
Junction Tree : Sum-Product Algorithm

- We first randomly choose a root clique, followed by message passing:
    - Inward messages towards the root clique from the leaf cliques.
    - Outward messages from the root clique towards the leaf cliques.
- Message passing protocol: C_i is ready to pass message to a neighbour C_j when it has received messages from all neighbors except for C_j.

Junction Tree : Sum-Product Algorithm

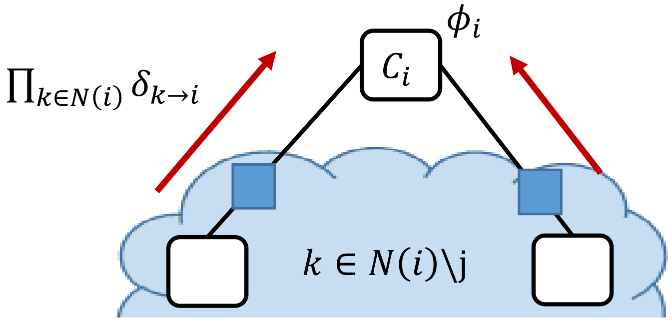
- Use the sum-product algorithm to compute messages from C_i to C_j:

$$\delta_{i\to j} = \sum_{C_i \setminus S_{ij}} \phi_i . \prod_{k\in N(i)\setminus j} \delta_{k\to i}$$



- The unnormalized* marginal probability of clique C_i is given by:

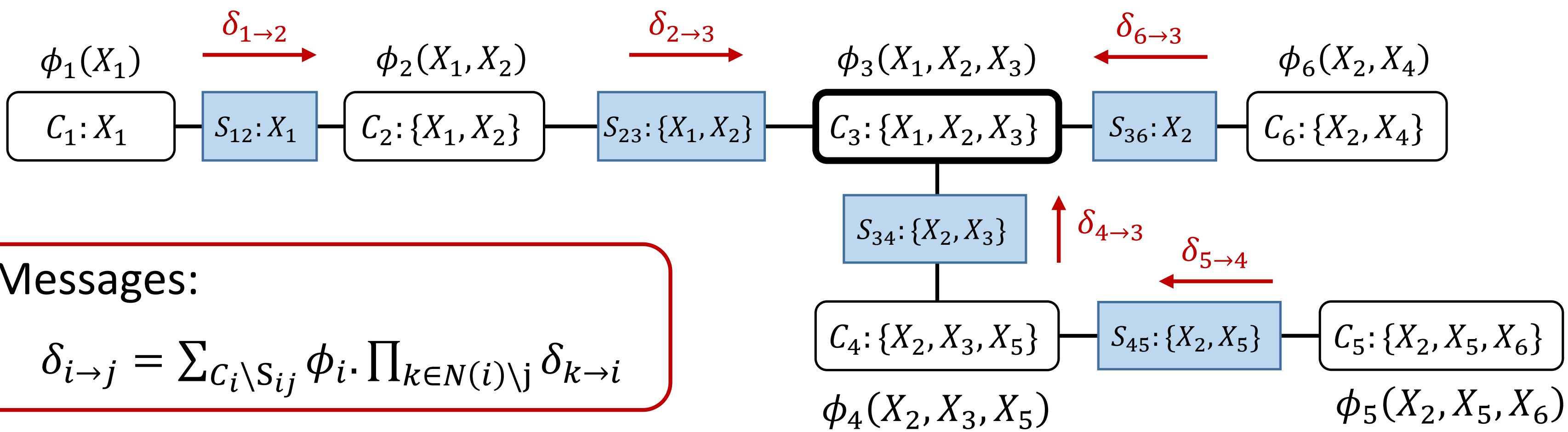
$$\tilde{p}(C_i) = \phi_i . \prod_{k\in N(i)} \delta_{k\to i}$$



*Unnormalized probability because the clique potentials come from the UGM potentials, where we ignored the partition function

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Messages:

$$\delta_{i\to j} = \sum_{C_i \setminus S_{ij}} \phi_i . \prod_{k\in N(i)\setminus j} \delta_{k\to i}$$

Inward pass:

$$\delta_{1\to2} = \sum_{C_1\setminus S_{12}} \phi_1 = \phi_1$$

$$\delta_{2\to3} = \sum_{C_2\setminus S_{23}} \phi_2 \,.\, \delta_{1\to2} = \phi_2 . \phi_1$$

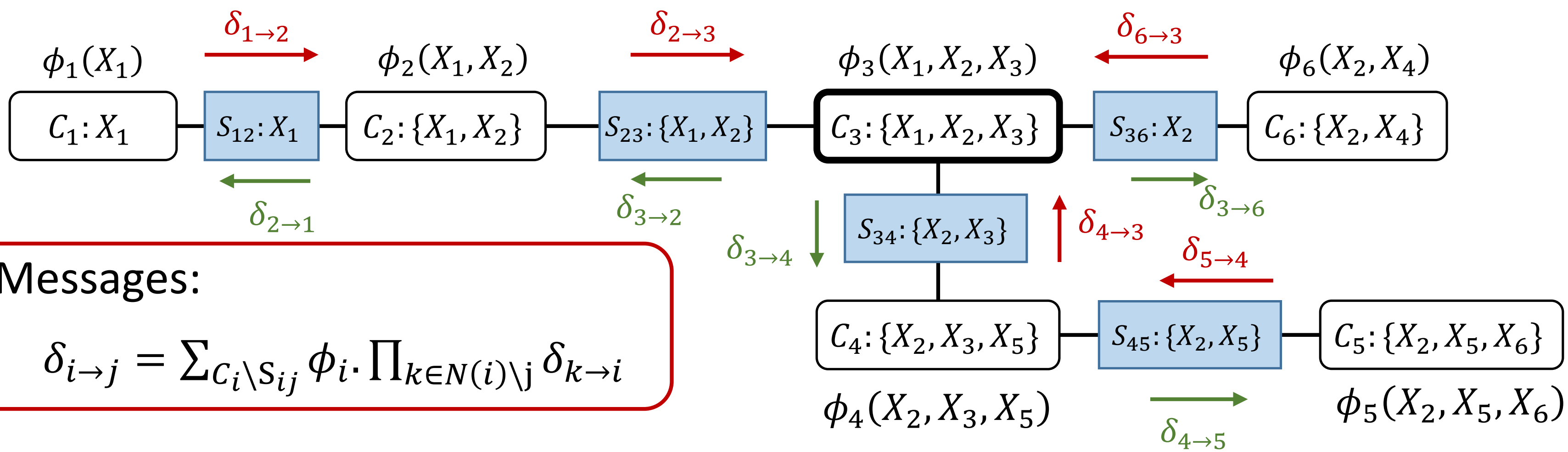
$$\delta_{5\to4} = \sum_{C_5\setminus S_{45}} \phi_5 = \sum_{X_6} \phi_5$$

$$\delta_{4\to3} = \sum_{C_4\setminus S_{34}} \phi_4 \,.\, \delta_{5\to3} = \sum_{X_5} \phi_4 \sum_{X_6} \phi_5$$

$$\delta_{6\to3} = \sum_{C_6\setminus S_{36}} \phi_6 = \sum_{X_4} \phi_6$$

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Messages:

$$\delta_{i\to j} = \sum_{C_i \setminus S_{ij}} \phi_i . \prod_{k\in N(i)\setminus j} \delta_{k\to i}$$

Inward pass:

$$\delta_{1\to2} = \sum_{C_1\setminus S_{12}} \phi_1 = \phi_1$$

$$\delta_{2\to3} = \sum_{C_2\setminus S_{23}} \phi_2 . \delta_{1\to2} = \phi_2 . \phi_1$$

$$\delta_{5\to4} = \sum_{C_5\setminus S_{45}} \phi_5 = \sum_{X_6} \phi_5$$

$$\delta_{4\to3} = \sum_{C_4\setminus S_{34}} \phi_4 . \delta_{5\to3} = \sum_{X_5} \phi_4 \sum_{X_6} \phi_5$$

$$\delta_{6\to3} = \sum_{C_6\setminus S_{36}} \phi_6 = \sum_{X_4} \phi_6$$

Outward pass:

$$\delta_{3\to2} = \sum_{C_3\setminus S_{23}} \phi_3 . \delta_{6\to3} . \delta_{4\to3}$$

$$\delta_{2\to1} = \sum_{C_2\setminus S_{12}} \phi_2 . \delta_{3\to2}$$

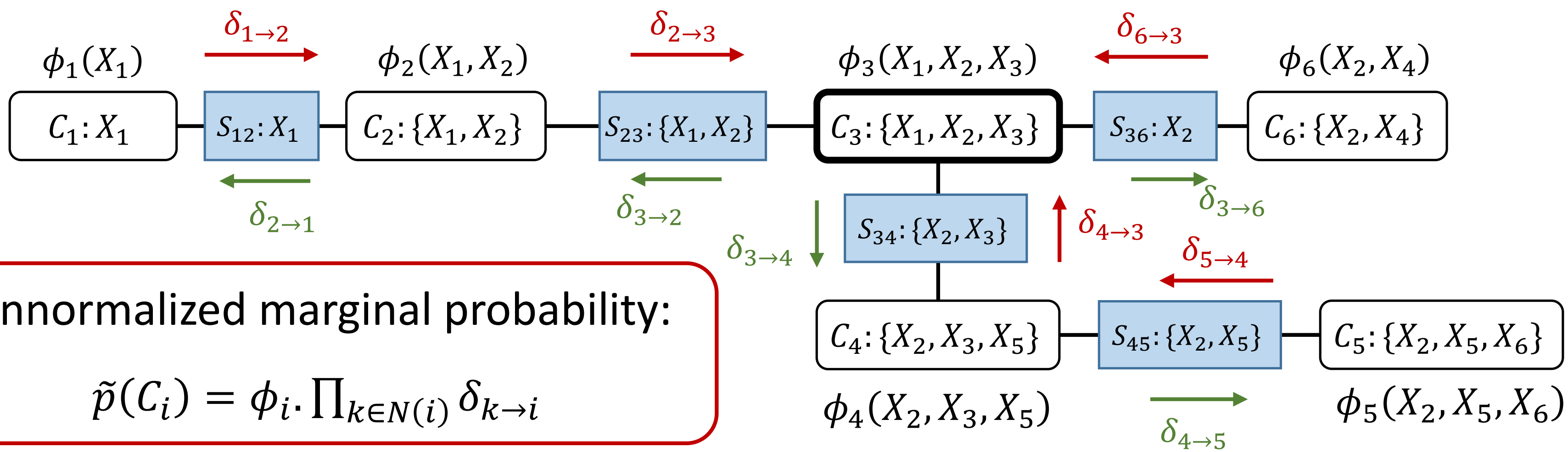
$$\delta_{3\to6} = \sum_{C_3\setminus S_{36}} \phi_3 . \delta_{2\to3} . \delta_{4\to3}$$

$$\delta_{3\to4} = \sum_{C_3\setminus S_{34}} \phi_3 . \delta_{2\to3} . \delta_{6\to3}$$

$$\delta_{4\to5} = \sum_{C_4\setminus S_{45}} \phi_4 . \delta_{3\to4}$$

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Unnormalized marginal probability:

$$\tilde{p}(C_i) = \phi_i . \prod_{k\in N(i)} \delta_{k\to i}$$

$$\begin{aligned}
\tilde{p}(C_1) = \tilde{p}(X_1) &= \phi_1 . \prod_{k\in N(1)} \delta_{k\to1} \\
&= \phi_1 . \delta_{2\to1} \\
&= \phi_1 . \sum_{C_2 \setminus S_{12}} \phi_2 \,. \delta_{3\to2} \\
&= \phi_1 . \sum_{X_2} \phi_2 \,. \sum_{C_3 \setminus S_{23}} \phi_3 \,. \delta_{6\to3} . \delta_{4\to3} \\
&= \phi_1 . \sum_{X_2} \phi_2 \,. \sum_{X_3} \phi_3 \,. \sum_{X_4} \phi_6 \;. \sum_{X_5} \phi_4 \;\; \sum_{X_6} \phi_5
\end{aligned}$$

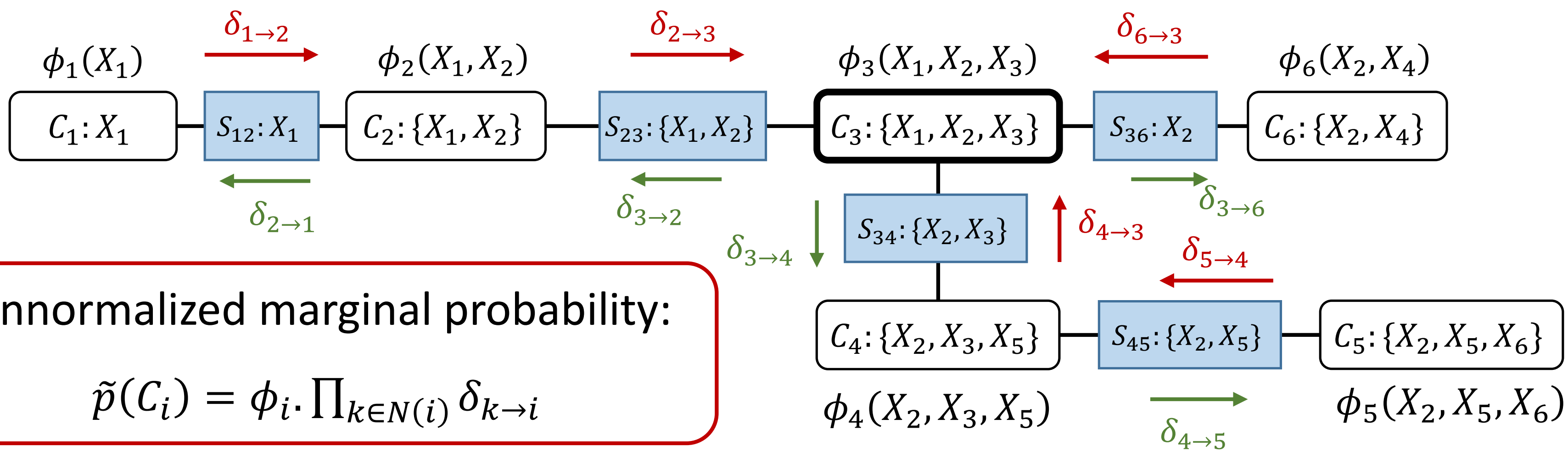
Marginal probability:

$$p(X_1) = \frac{\tilde{p}(X_1)}{\sum_{X_1} \tilde{p}(X_1)}$$

Result is equivalent to variable elimination!

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Unnormalized marginal probability:

$$\tilde{p}(C_i) = \phi_i . \prod_{k \in N(i)} \delta_{k\to i}$$

$$\tilde{p}(C_2) = \tilde{p}(X_1, X_2)$$

$$= \phi_2 . \prod_{k \in N(2)} \delta_{k\to2}$$

$$= \phi_2 . \delta_{1\to2} \,.\, \delta_{3\to2}$$

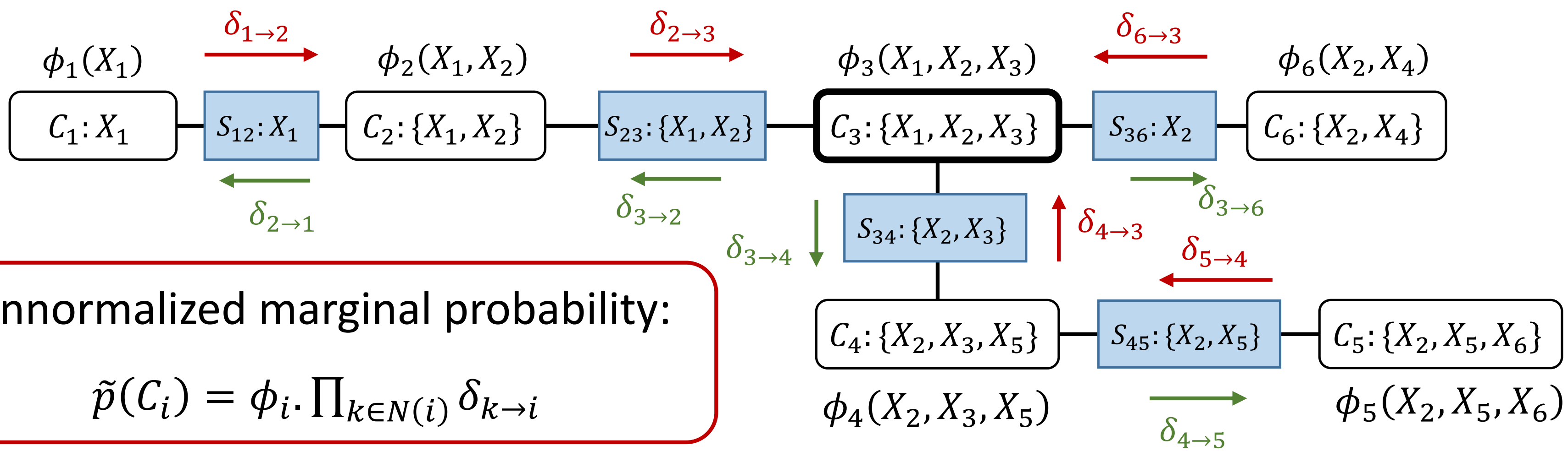
Marginal probabilities:

$$p(X_1, X_2) = \frac{\tilde{p}(X_1, X_2)}{\sum_{X_1} \sum_{X_2} \tilde{p}(X_1, X_2)}$$

$$p(X_2) = \sum_{X_1} p(X_1, X_2)$$

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Unnormalized marginal probability:

$$\tilde{p}(C_i) = \phi_i . \prod_{k \in N(i)} \delta_{k\to i}$$

$$\tilde{p}(C_3) = \tilde{p}(X_1, X_2, X_3)$$
$$= \phi_3 . \delta_{2\to3} . \delta_{6\to3} . \delta_{4\to3}$$

$$\tilde{p}(C_4) = \tilde{p}(X_2, X_3, X_5)$$
$$= \phi_4 . \delta_{3\to4} . \delta_{5\to4}$$

$$\tilde{p}(C_5) = \tilde{p}(X_2, X_5, X_6)$$
$$= \phi_5 . \delta_{4\to5}$$

$$\tilde{p}(C_6) = \tilde{p}(X_2, X_4)$$
$$= \phi_6 . \delta_{3\to6}$$

Constructing the Junction Tree

1. Triangulation: Get the reconstituted graph

Choose an elimination ordering I

$\textsc{DirectedGraphEliminate}(G, I)$

1. $G^m = \textsc{Moralize}(G)$ // for DGM, skip this step if UGM
2. $\textsc{UndirectedGraphEliminate}(G^m, I)$ // get reconstituted graph

1. $\textsc{Moralize}(G)$

for each node X_i in I
    connect all of the parents of X_i
end drop the orientation of all edges
return G

2. $\textsc{UndirectedGraphEliminate}(\mathcal{G}, I)$

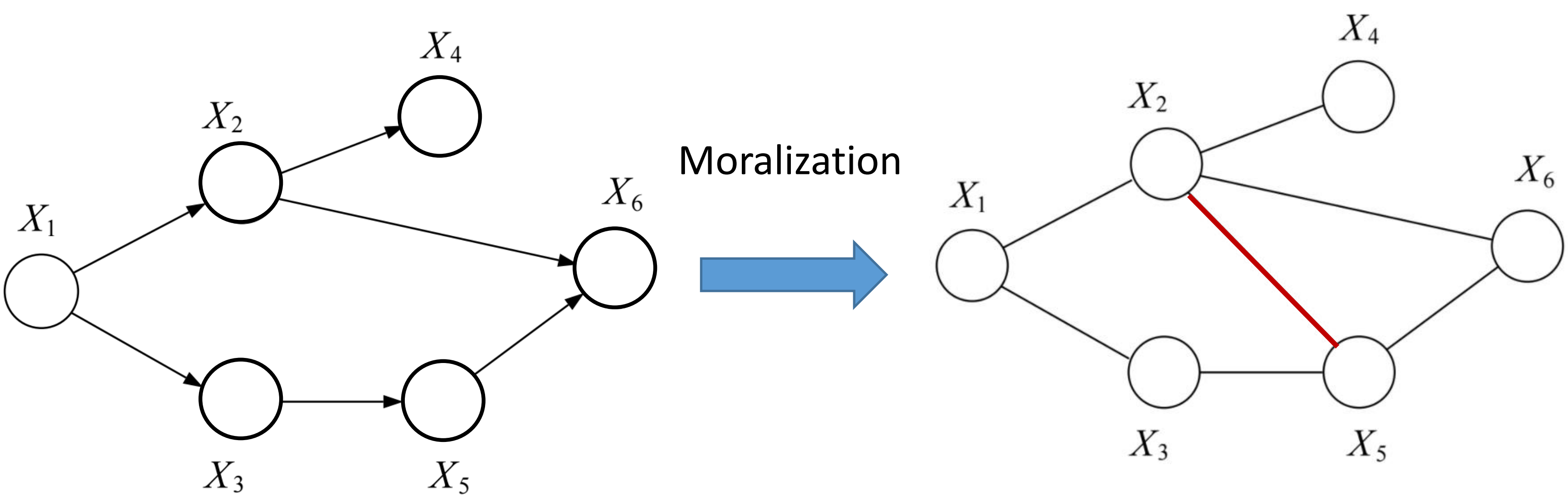
for each node X_i in I
    connect all of the remaining neighbors of X_i
    remove X_i from the graph
end

Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Constructing the Junction Tree

1. Triangulation: Get the reconstituted graph

Choose an elimination ordering $I = (6; 5; 4; 3; 2; 1)$



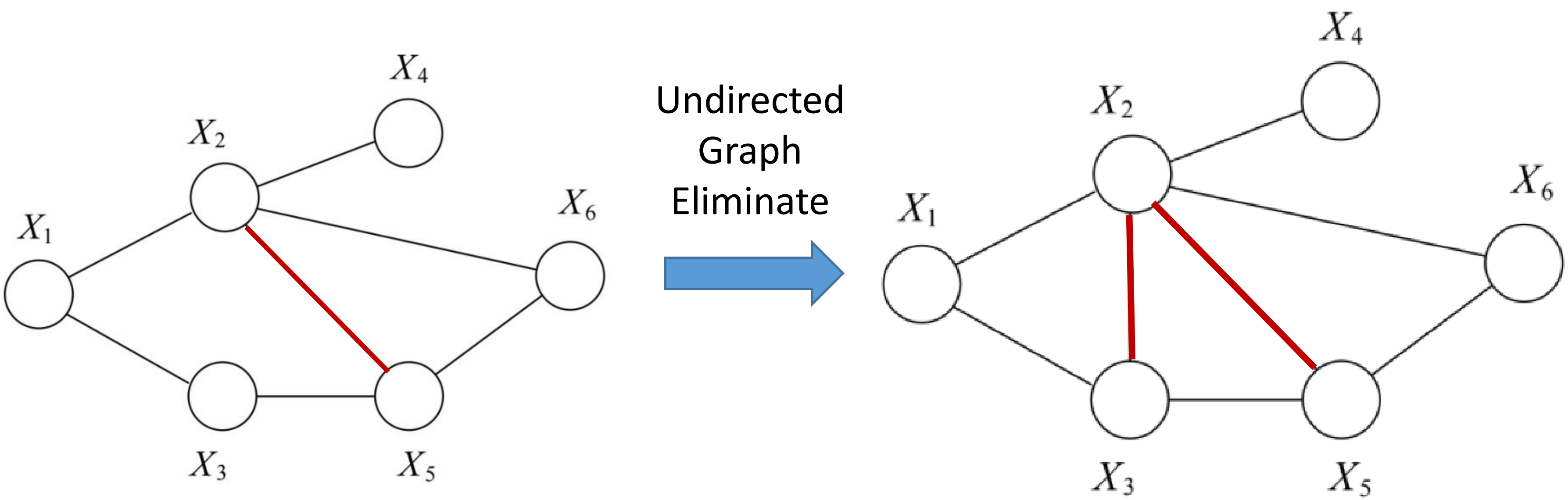
Parents are “married”

Source: “An introduction to probabilistic graphical models”, Michael I. Jordan, 2002.

Constructing the Junction Tree

1. Triangulation: Get the reconstituted graph

Choose an elimination ordering $I = (6; 5; 4; 3; 2; 1)$



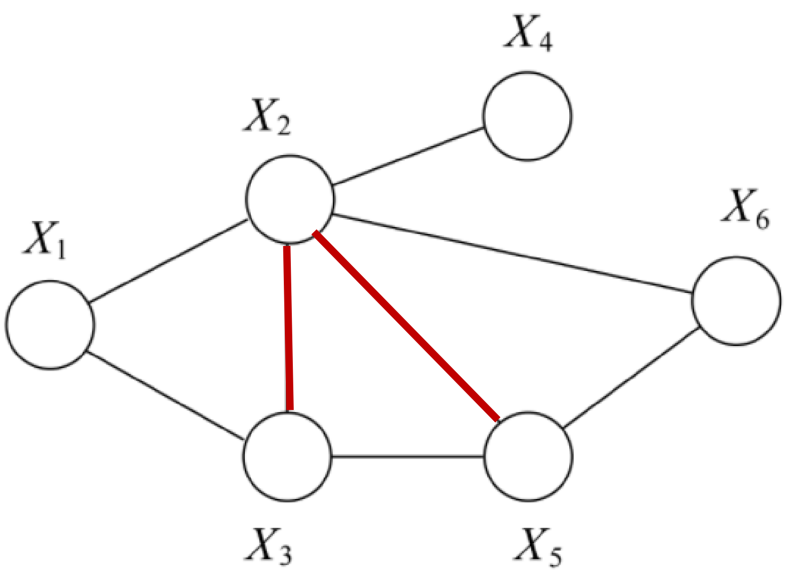
Parents are "married"

Reconstituted graph: additional edges (red) added during the elimination process

Image source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Constructing the Junction Tree

2. Get all clusters and all possible sepsets: Use eliminate cliques as clusters, a possible sepset is $S_{ij} = C_i \cap C_j$.



$C_5: \{X_2, X_5, X_6\}$

$C_4: \{X_2, X_3, X_5\}$

$C_6: \{X_2, X_4\}$

$C_3: \{X_1, X_2, X_3\}$

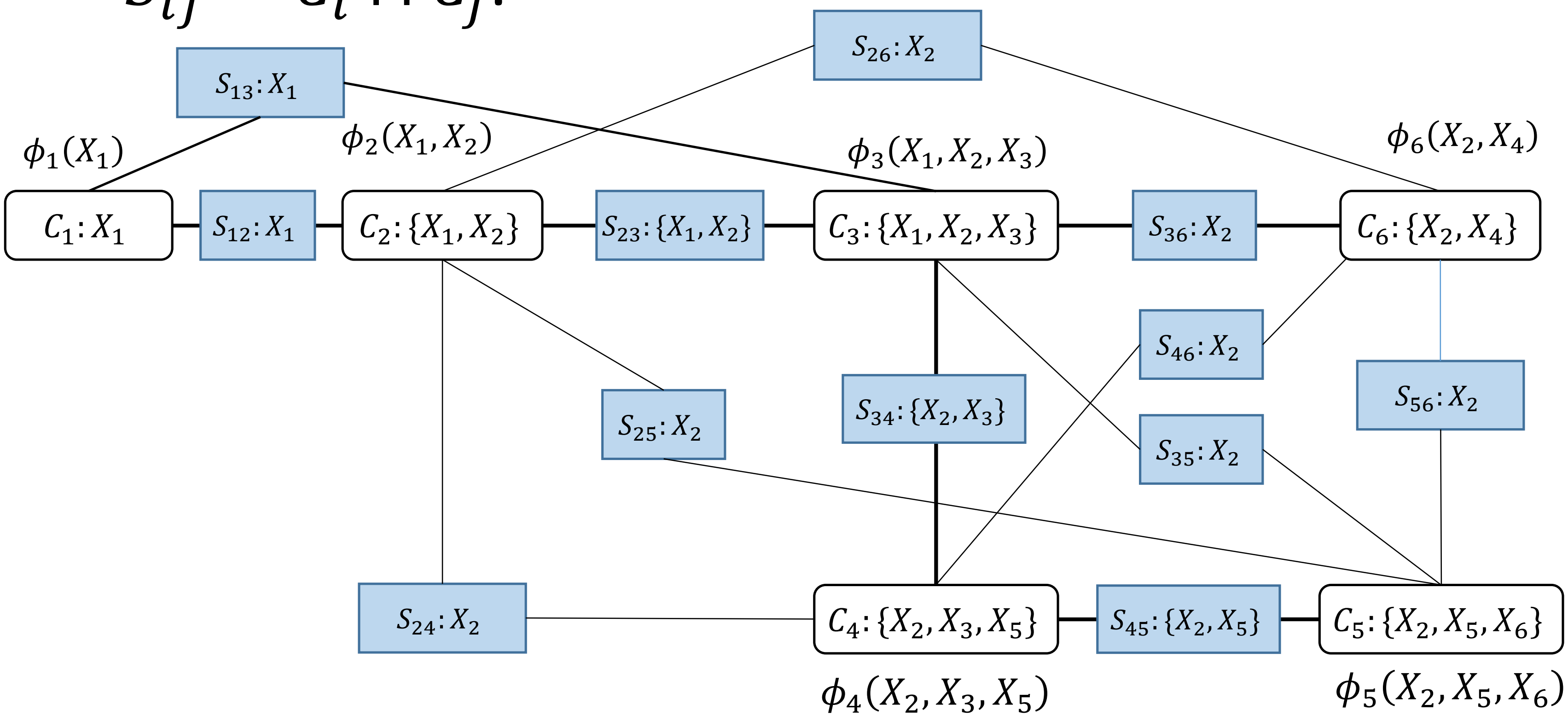
$C_2: \{X_1, X_2\}$

$C_1: X_1$

Image source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Constructing the Junction Tree

2. Get all clusters and all possible sepsets: Use eliminate cliques as clusters, a possible sepset is $S_{ij} = C_i \cap C_j$.

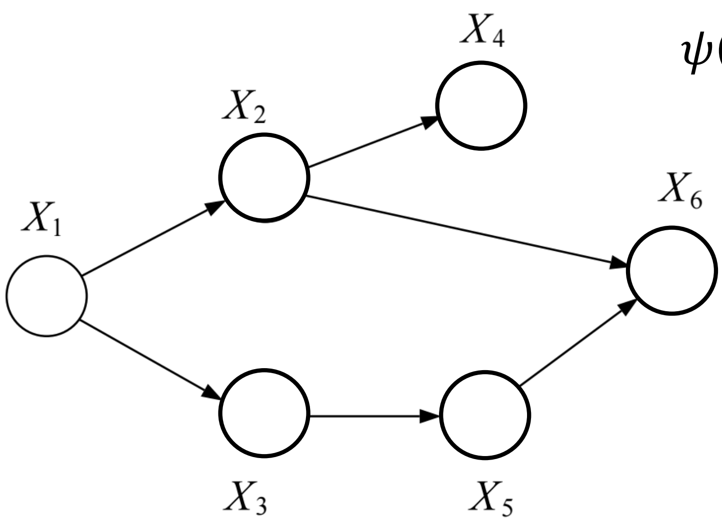


Constructing the Junction Tree

3. Assign cluster potentials: cluster potentials are formed by condition probabilities (DGM), or potentials (UGM).

$$p(\mathrm{x}_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2,x_5)$$

$$\psi(\mathrm{x}_1)\psi(x_1,x_2)\psi(x_1,x_3)\psi(x_2,x_4)\psi(x_3,x_5)\psi(x_2,x_5,x_6)$$



Use each conditional probability / potential only once!

$\phi_1(X_1) = p(x_1)$, $\phi_2(X_1, X_2) = p(x_2|x_1)$

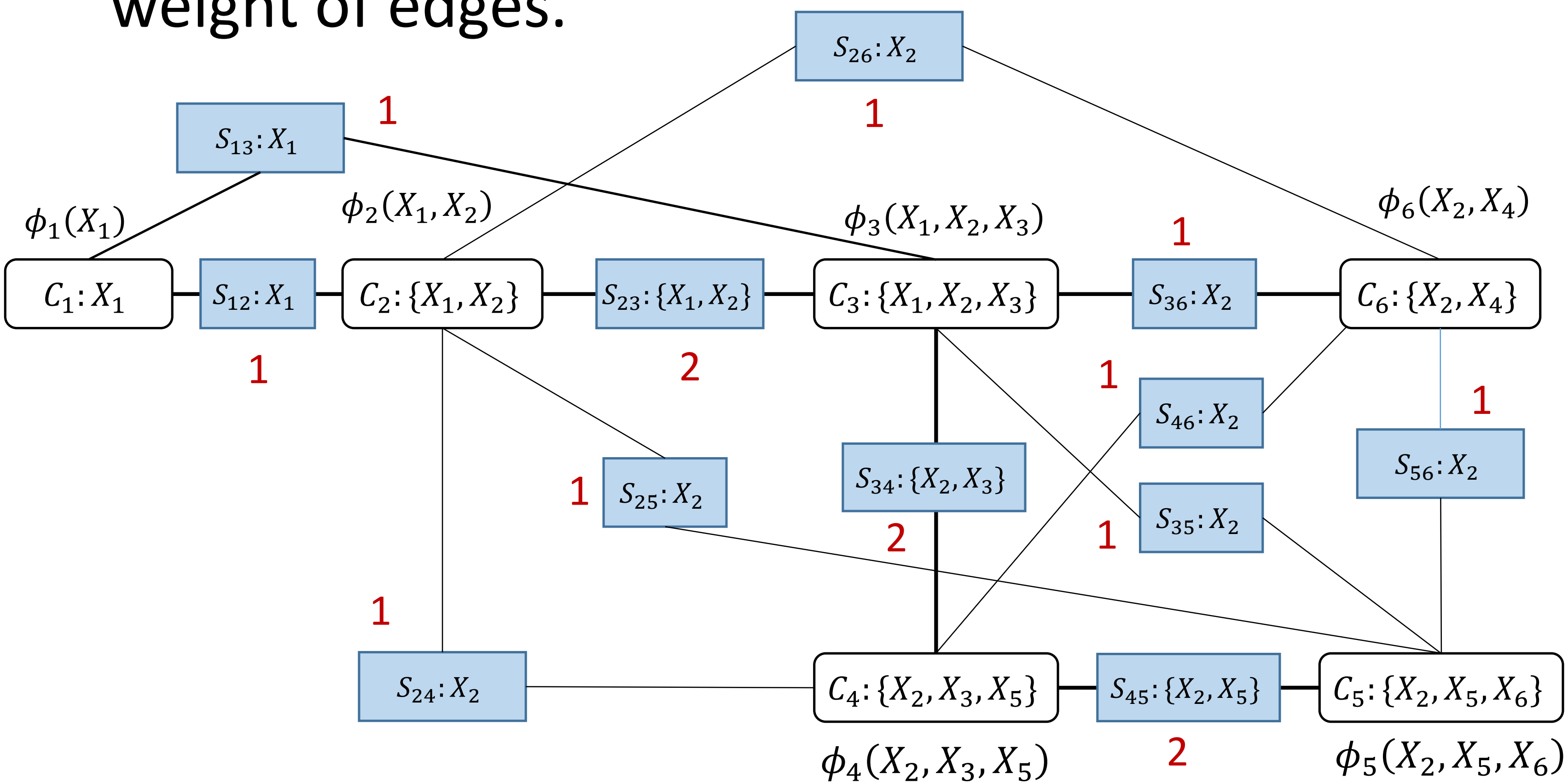
$\phi_3(X_1, X_2, X_3) = p(x_3|x_1)$, $\phi_4(X_2, X_3, X_5) = p(x_5|x_3)$

$\phi_5(X_2, X_5, X_6) = p(x_6|x_2, x_5)$,

$\phi_6(X_2, X_4) = p(x_4|x_2)$

Constructing the Junction Tree

3. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.



Constructing the Junction Tree

3. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.

Theorem: A cluster tree T is a clique tree / junction tree only if it is a maximal spanning tree.

Constructing the Junction Tree

Proof:

Consider a random variable X_k and a cluster tree T with cluster C_i and sepset S_j, the fact that T is a tree implies:

$1(a)$: indicator function that returns 1 if a is true, 0 otherwise

Minus 1 because for tree #edges = #nodes -1

$$\underbrace{\sum_{j=1}^{M-1} 1(X_k \in S_j)}_{\text{\# times } X_k \text{ appear in the sepsets}} \leq \underbrace{\sum_{i=1}^{M} 1(X_k \in C_i)}_{\text{\# times } X_k \text{ appear in the cluster}} - 1,$$

M: # clusters

The inequality sign becomes equality when X_k forms a sub-tree, i.e. running intersection property is fulfilled.

Constructing the Junction Tree

Proof:

Total weight of a cluster tree $w(T)$ is equal to the sum of the cardinalities of its sepsets:

$$
\begin{aligned}
w(T) &= \sum_{j=1}^{M-1} |S_j| \\
&= \sum_{j=1}^{M-1} \sum_{k=1}^{N} 1(X_k \in S_j) \\
&= \underbrace{\sum_{k=1}^{N} \sum_{j=1}^{M-1} 1(X_k \in S_j)}_{\text{sum of cardinalities of all sepsets}} \leq \overbrace{\sum_{k=1}^{N} \left[\sum_{i=1}^{M} 1(X_k \in C_i) - 1 \right]}^{\text{sum of cardinalities of all clusters minus \# random variables}} \\
&= \sum_{i=1}^{M} \sum_{k=1}^{N} 1(X_k \in C_i) - N \\
&= \sum_{i=1}^{M} |C_i| - N
\end{aligned}
$$

From the previous slide

M: # cliques
N: # random variables

Constructing the Junction Tree

Proof:

$$w(T) \;=\; \sum_{j=1}^{M-1} |S_j| \;\leq\; \sum_{k=1}^{N} \left[\sum_{i=1}^{M} 1(X_k \in C_i) - 1 \right]$$

M: # cliques
N: # random variables

- We saw from previous slide that for the running intersection property, i.e. junction tree to hold, the inequality has to become equality.
- This implies a maximum sum of cardinalities of all sepsets, i.e. maximal spanning tree!

Constructing the Junction Tree

3. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(u, v)
return A
```

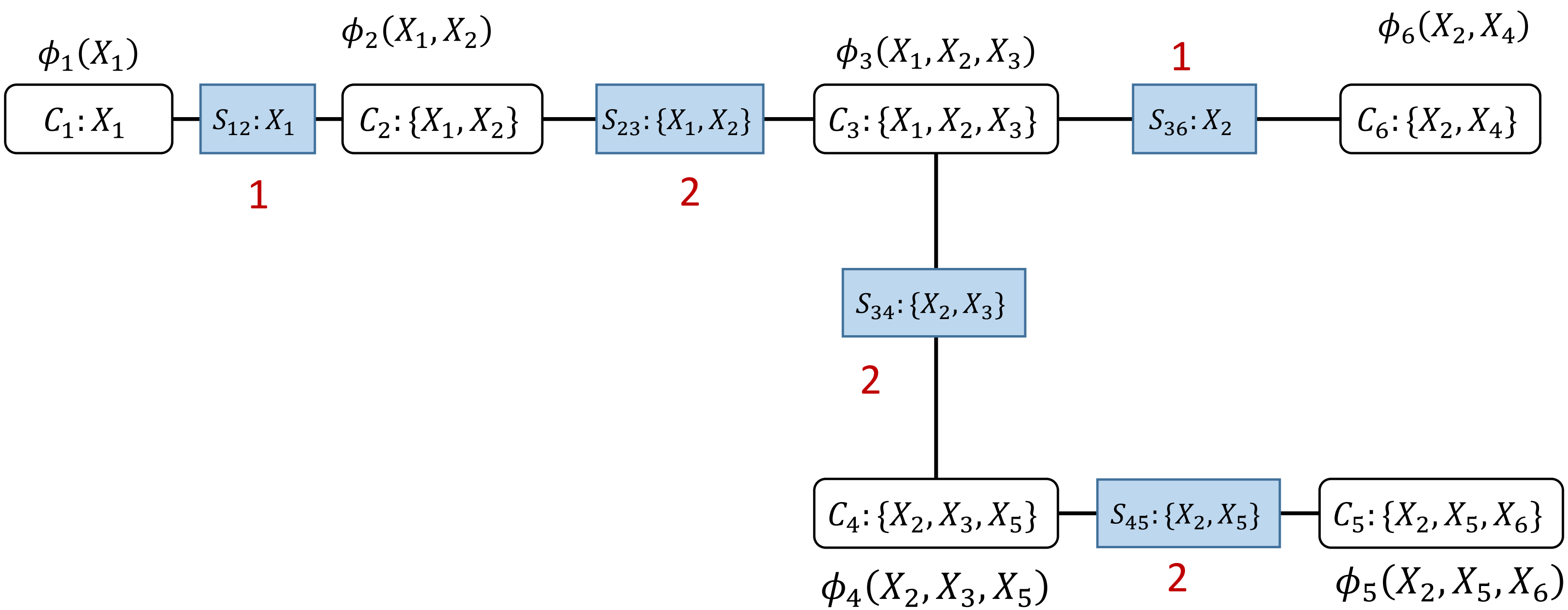
Can be more than 1 maximum spanning tree!

Source: https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Constructing the Junction Tree

3. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.

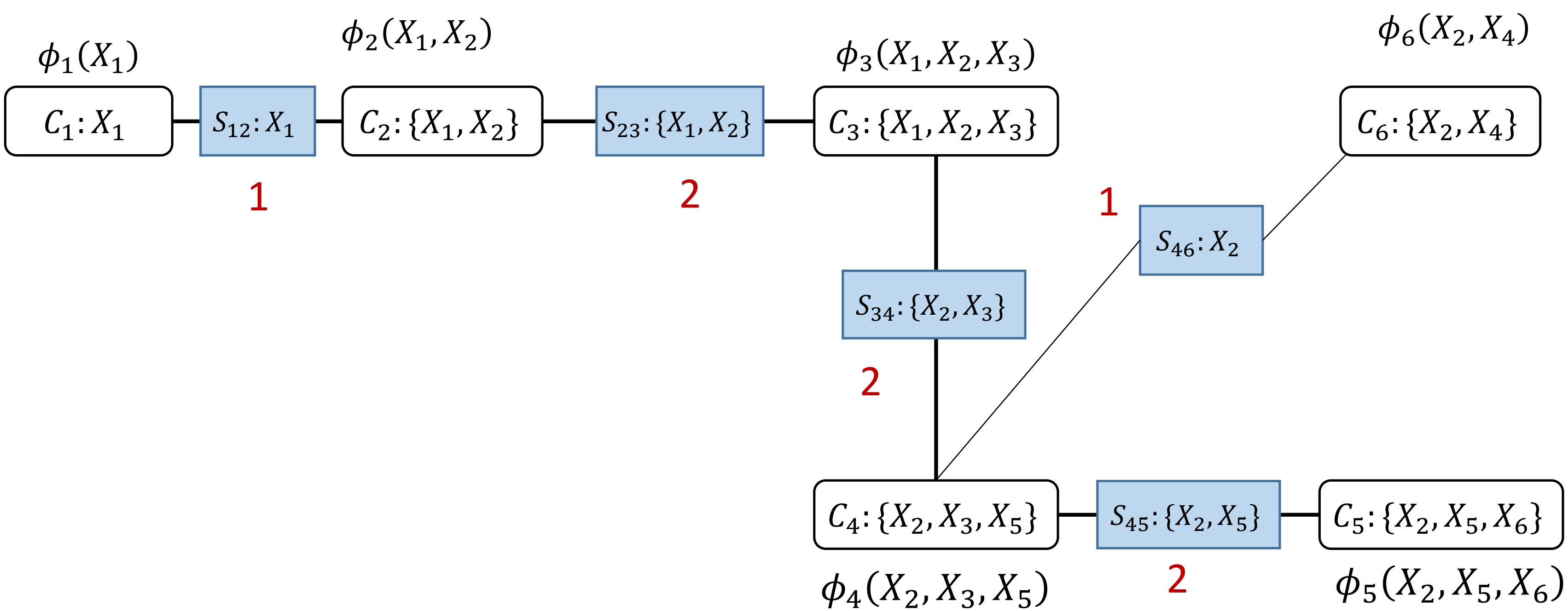
Example:



Constructing the Junction Tree

3. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.

Example:



Summary

- We have looked at how to:

1. Represent a joint distribution with a factor graph, and use it to compute the marginal/conditional probabilities.

2. Use the max-product algorithm to find the maximal probability and its configurations.

3. Convert a DGM/UGM into the junction tree and use it to compute the marginal/conditional probabilities.