NUS National University of Singapore | School of Computing

CS5340 Uncertainty Modeling in AI

Lecture 6: Factor Graph and the Junction Tree Algorithm

Asst. Prof. Harold Soh

AY 2022/23

Semester 2

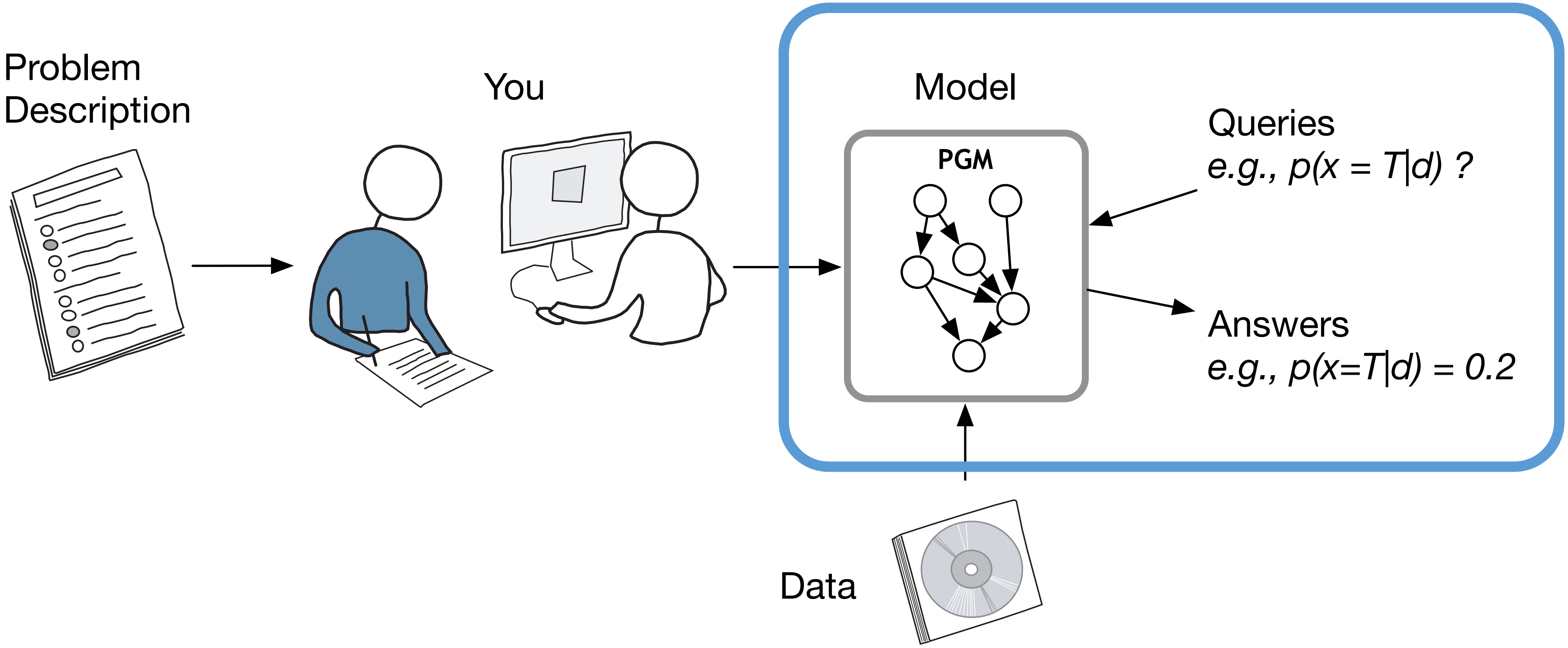
NUS National University of Singapore | School of Computing

Recap from Lecture 5

Variable Elimination and Belief Propagation

CS5340 in a nutshell

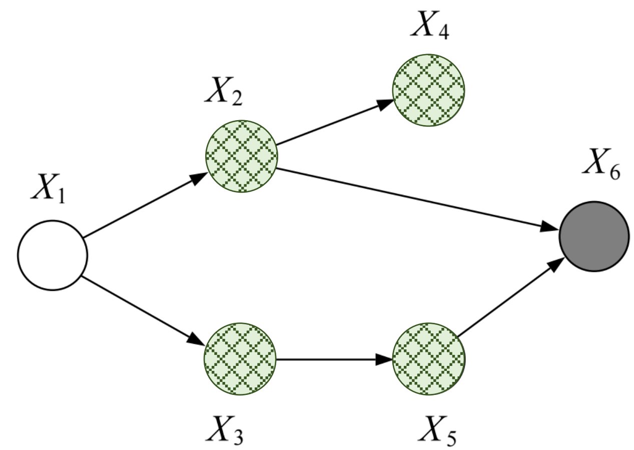
CS5340 is about how to "**represent**" and "**reason**" with uncertainty in a computer.



Variable Elimination

Conditional probability:

$$p(x_1|\bar{x}_6) = \frac{p(x_1, \bar{x}_6)}{p(\bar{x}_6)}$$



Marginal probability:

$$\begin{aligned} p(x_1, \bar{x}_6) &= \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1)p(x_4 \mid x_2)p(x_5 \mid x_3)p(\bar{x}_6 \mid x_2, x_5) \\ &= p(x_1)\sum_{x_2} p(x_2 \mid x_1)\sum_{x_3} p(x_3 \mid x_1)\sum_{x_4} p(x_4 \mid x_2)\underbrace{\sum_{x_5} p(x_5 \mid x_3)p(\bar{x}_6 \mid x_2, x_5)}_{m_5(x_2, x_3)} \end{aligned}$$

eliminate X_5

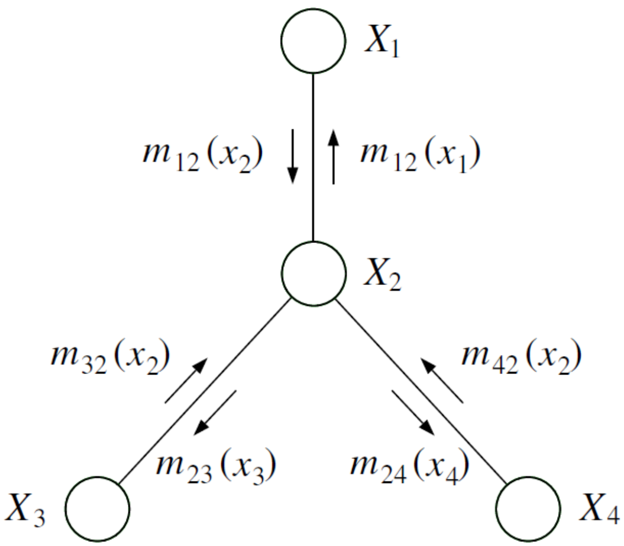
- Summands can be pushed in due to the distributive law.
- $m_i(x_{S_i})$ denote the expression from performing Σ_{x_i}, where X_{S_i} are the variables, other than X_i, that appear in the summand.

Sum-Product Algorithm

- Two phases:

1. Messages flow inward from leaves toward the root.

2. Initiated once all incoming messages have been received by the root node – messages flow outward from root toward the leaves.

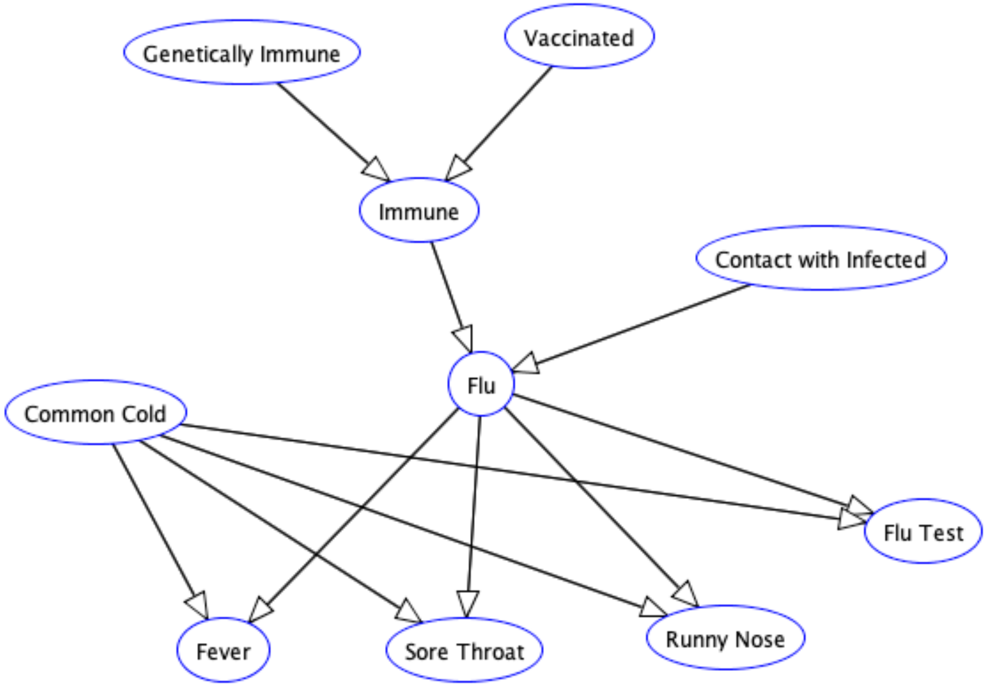


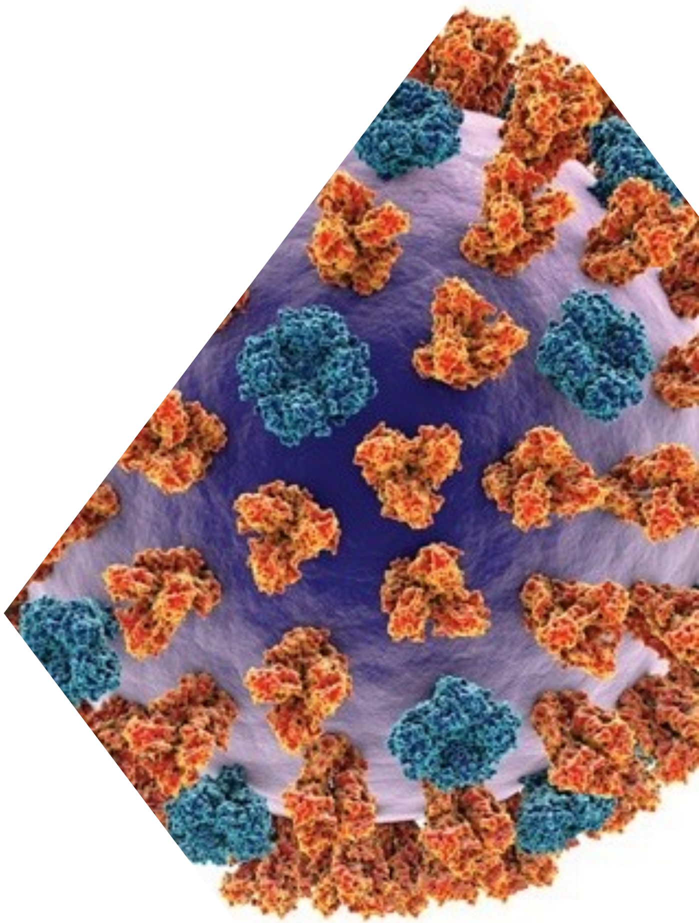
Ideas Summary

- Variable Elimination works for all graphs, but is query specific and can be computationally expensive.
- For *Trees*, use Sum-Product (Belief Propagation) algorithm which is very efficient.

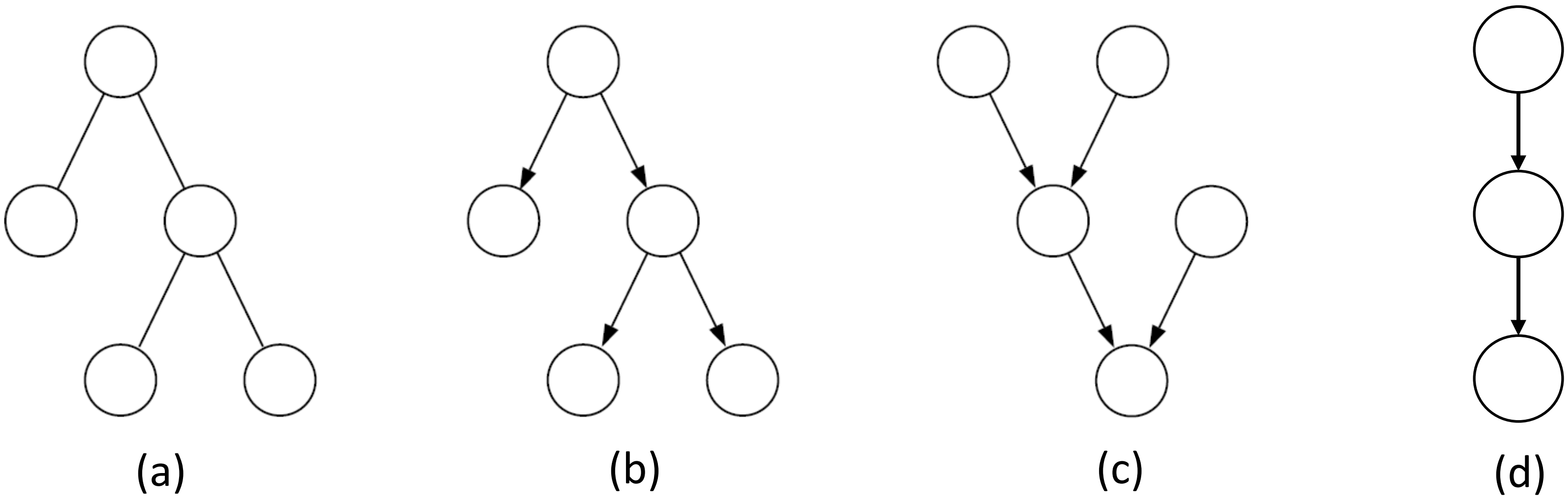
what if we want to address more complex structures than trees?

Our flu example was not a tree!





"Tree-Like" Graphs



a) Undirected tree: without any loop (unique path between any two nodes)

b) Directed tree: only 1 single parent for every node, moralizations lead to an undirected tree.

c) Polytree: nodes with more than 1 parent. Not a directed tree, moralizations lead to loops.

d) Chain: this is also a directed tree (more on chains when we look at Hidden Markov Models).

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

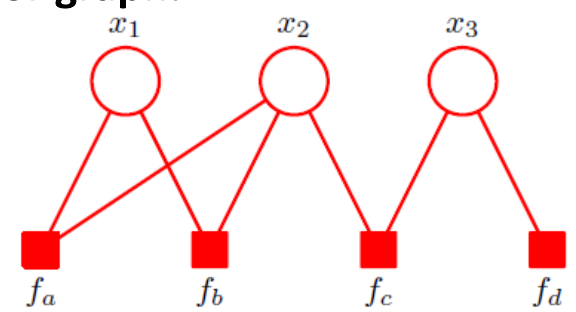
Course Schedule

Week	Date	Lecture Topic	Tutorial Topic
1	12 Jan	Introduction to Uncertainty Modeling + Probability Basics	Introduction
2	19 Jan	Simple Probabilistic Models	Probability Basics
3	26 Jan	Bayesian networks (Directed graphical models)	More Basic Probability
4	2 Feb	Markov random Fields (Undirected graphical models)	DGM modelling and d-separation
5	9 Feb	Variable elimination and belief propagation	MRF + Sum/Max Product
6	16 Feb	Factor graph and the junction tree algorithm	**Quiz 1**
-	-	RECESS WEEK	
7	2 Mar	Mixture Models and Expectation Maximization (EM)	Linear Gaussian Models
8	9 Mar	Hidden Markov Models (HMM)	Probabilistic PCA
9	16 Mar	Monte-Carlo Inference (Sampling)	Linear Gaussian Dynamical System
10	23 Mar	Variational Inference	MCMC + Sequential VAE
11	30 Mar	Inference and Decision-Making (Special Topic)	**Quiz 2**
12	6 Apr	Gaussian Processes (Special Topic)	Wellness Day
13	13 Apr	**Project Presentations**	Closing

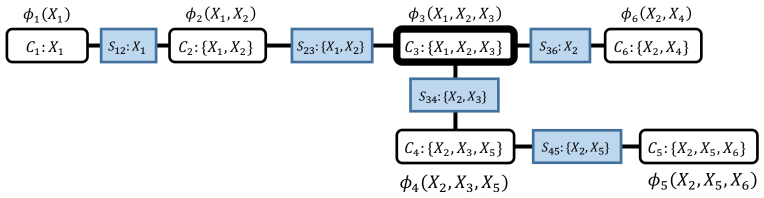
Today: Key ideas

- We will introduce 2 new data structures:
  - Factor graph
    - Works on polytrees
  - Junction tree
    - Works in general
- In both cases, we will learn a sum-product algorithm

Factor graph:



Junction tree:



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
6. Lee Gim Hee’s slides.

Learning Outcomes

- Students should be able to:

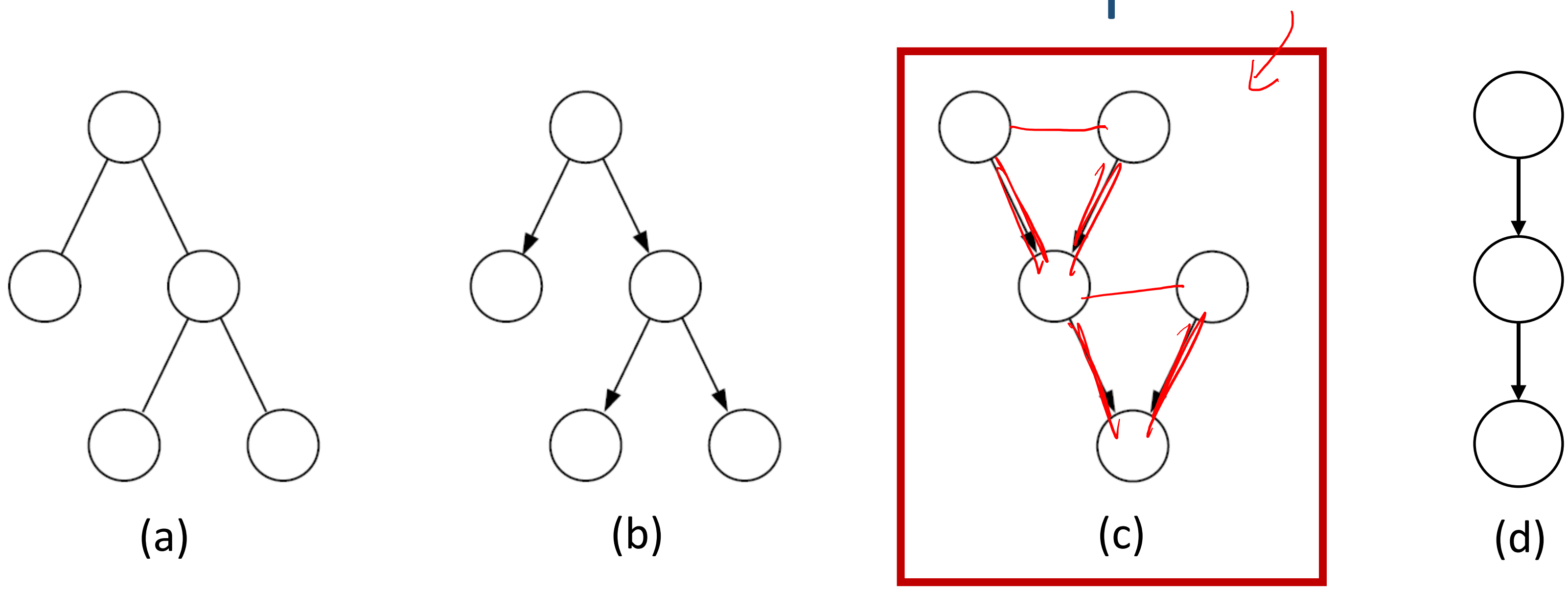
1. Represent a joint distribution with a factor graph, and use it to compute the marginal/conditional probabilities.

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

NUS National University of Singapore | School of Computing

Factor Graphs

"Tree-Like" Graphs



a) Undirected tree: without any loop (unique path between any two nodes)

b) Directed tree: only 1 single parent for every node, moralizations lead to an undirected tree.

c) Polytree: nodes with more than 1 parent. Not a directed tree, moralizations lead to loops.

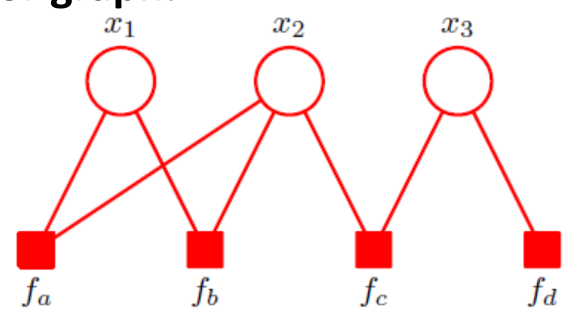
d) Chain: this is also a directed tree (more on chains when we look at Hidden Markov Models).

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

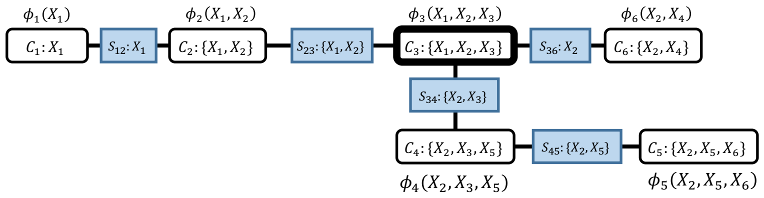
Today: Key ideas

- We will introduce 2 new data structures:
  - Factor graph
    - Works on polytrees
  - Junction tree
    - Works in general
- In both cases, we will learn a sum-product algorithm

Factor graph:

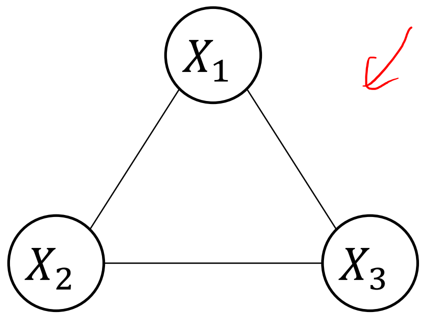


Junction tree:



Question: Factorization for UGM

- What is the factorization for this UGM?



A. $\phi(x_1, x_2)\phi(x_2, x_3)\phi(x_1, x_3)$

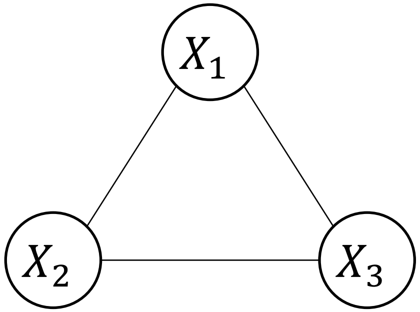
B. $\phi(x_1, x_2, x_3)$

C. $\phi(x_1)\phi(x_2)\phi(x_3)\phi(x_1, x_2, x_3)$

D. $\phi(x_1, x_2)\phi(x_2, x_3)\phi(x_1, x_3)\ \phi(x_1)\phi(x_2)\phi(x_3)$

Question: Factorization for UGM

- What is the factorization for this UGM?



All of them are right!

A. $\phi(x_1, x_2)\phi(x_2, x_3)\phi(x_1, x_3)$

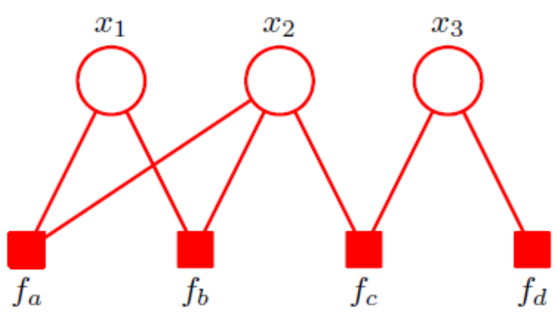
B. $\phi(x_1, x_2, x_3)$

C. $\phi(x_1)\phi(x_2)\phi(x_3)\phi(x_1, x_2, x_3)$

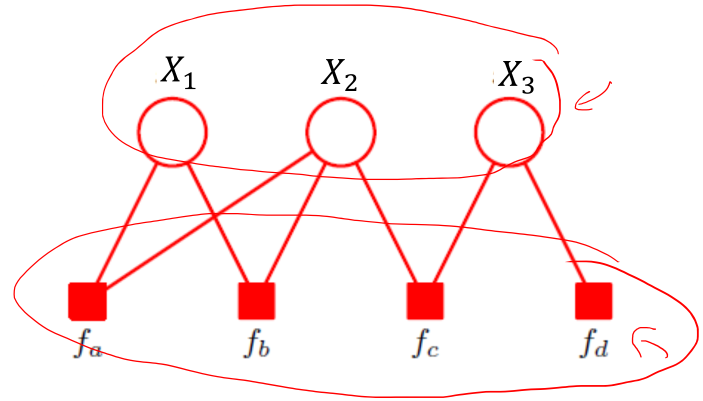
D. $\phi(x_1, x_2)\phi(x_2, x_3)\phi(x_1, x_3)\ \phi(x_1)\phi(x_2)\phi(x_3)$

Factor Graphs

- DGMs and UGMs: a global function of several variables is expressed as a product of factors over subsets of those variables.
- Factor graphs make this decomposition explicit by introducing additional nodes for the factors.



Factor Graphs: Graphical Representation



- A factor graph is a bipartite graph:

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

where

  - vertices $\mathcal{V} \in \{X_1, \ldots, X_n\}$: index the random variables,
  - vertices $\mathcal{F} \in \{\ldots, f_s, \ldots\}$: 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:

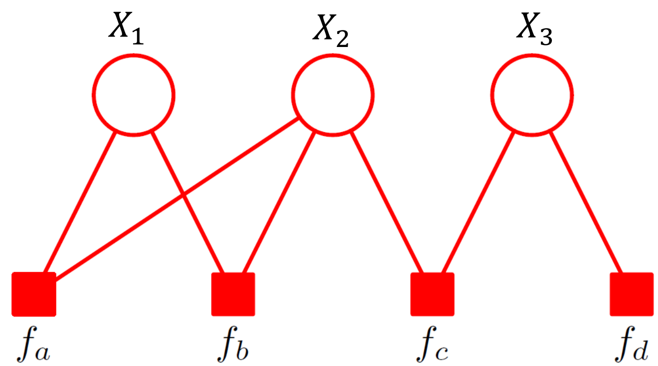
$$\boxed{p(\mathbf{x}) = \prod_{s} f_s(\mathbf{x}_s)}$$

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

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

Factor Graphs

Example:



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

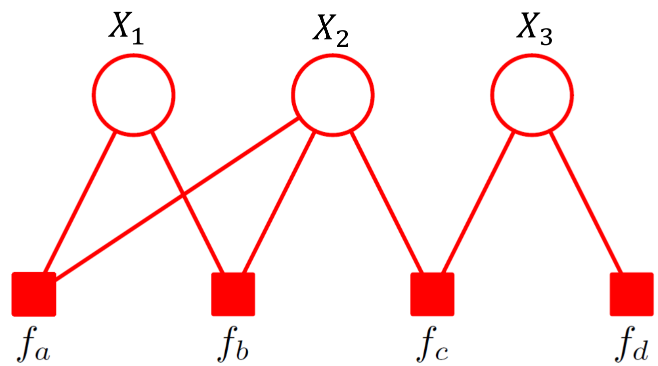
- Note that there are two factors $f_a(x_1, x_2)$ and $f_b(x_1, x_2)$ that are defined over the same set of variables.

- 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:



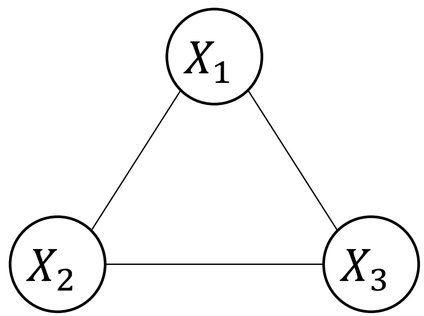
$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

- 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.

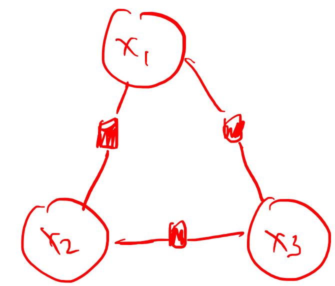
Image source: “Pattern recognition and machine learning”, Christopher Bishop

Question: Factorization for UGM

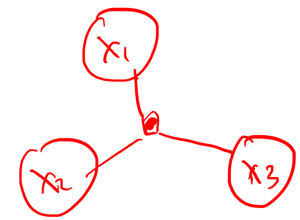
- Draw out the factor graphs for each of the factorizations below.



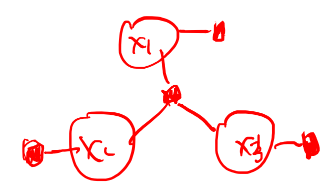
A)



B)



C)



A. $\phi(x_1, x_2)\phi(x_2, x_3)\phi(x_1, x_3)$
B. $\phi(x_1, x_2, x_3)$
C. $\phi(x_1)\phi(x_2)\phi(x_3)\phi(x_1, x_2, x_3)$
D. $\phi(x_1, x_2)\phi(x_2, x_3)\phi(x_1, x_3)\ \phi(x_1)\phi(x_2)\phi(x_3)$

Convert DGM to Factor Graph

- Recall the factorization of DGMs is defined as:

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

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

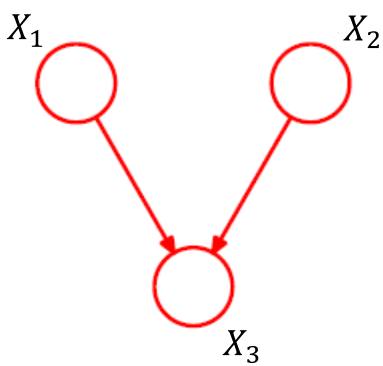
- Normalizing coefficient $1/Z$ can be viewed as a factor defined over the empty set of variables.

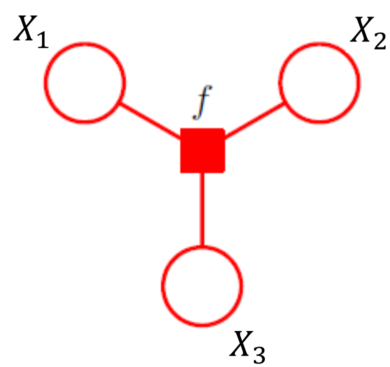
DGM/UGM to Factor Graph

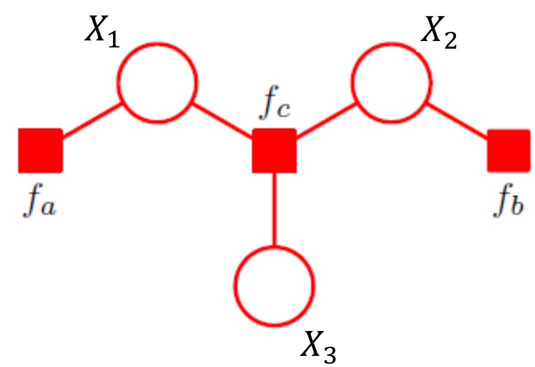
- Note that there may be several different factor graphs that correspond to the same DGM / UGM.
- Factor graphs are more specific about the precise form of the factorization.

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

Example: Directed Graph







$p(x_1)p(x_2)p(x_3|x_1, x_2)$

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

Two factor graphs representing the same distribution

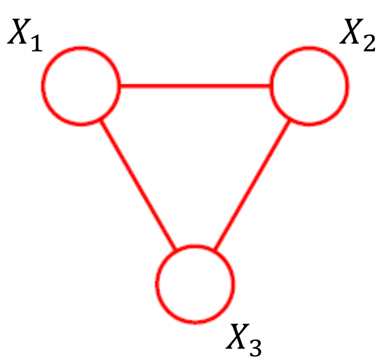
Image source: “Pattern recognition and machine learning”, Christopher Bishop

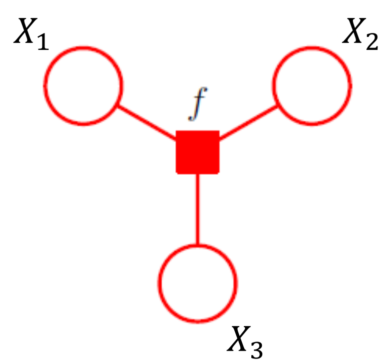
DGM/UGM to Factor Graph

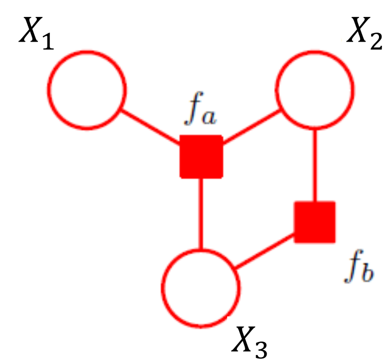
- Note that there may be several different factor graphs that correspond to the same DGM / UGM.
- Factor graphs are more specific about the precise form of the factorization.

Example: Undirected Graph

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







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

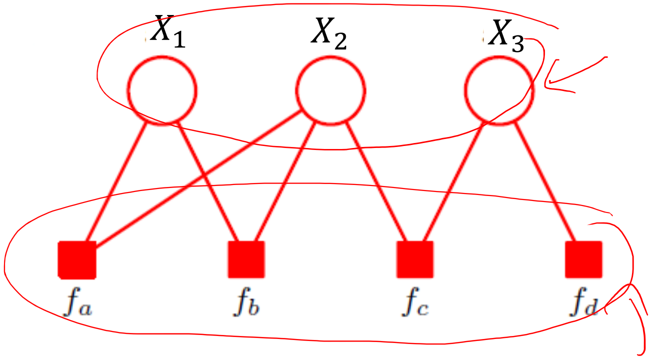
Image source: “Pattern recognition and machine learning”, Christopher Bishop

Factor Graphs and Sum Product

Belief Propagation on Factor Graphs

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

Key Idea: Tree after conversion to Factor Graphs

- 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

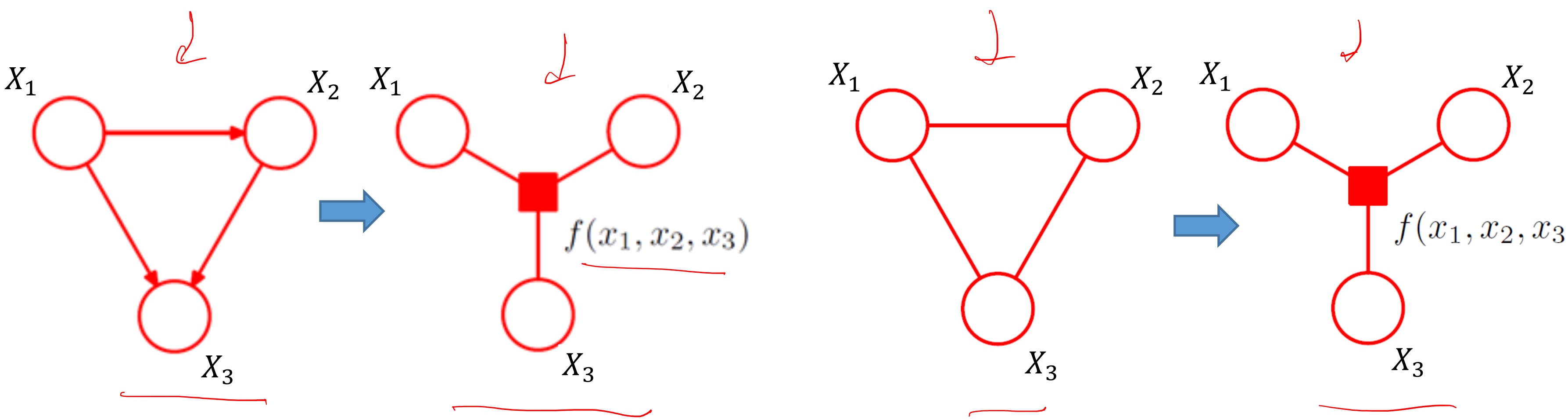
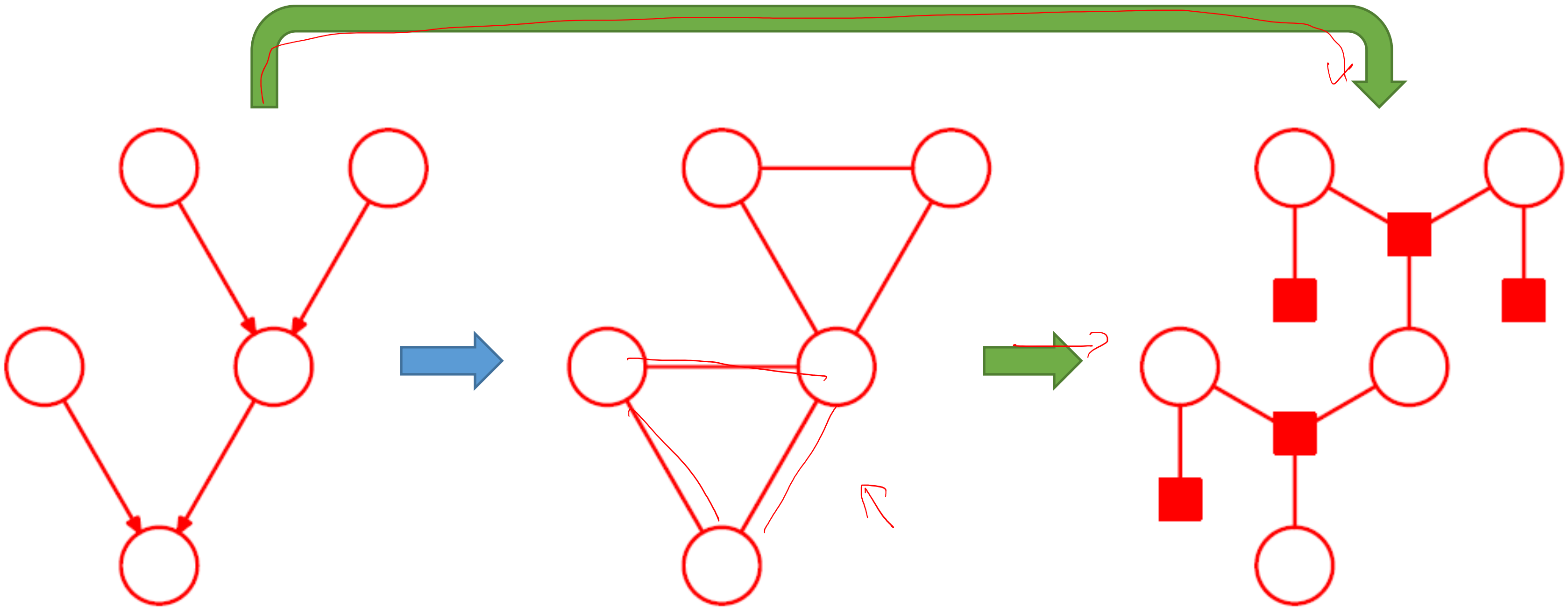


Image source: "Pattern recognition and machine learning", Christopher Bishop

Polytrees



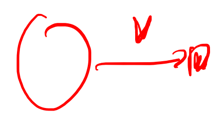
- Cycles appear after directed to undirected graph conversion.
- Local cycles disappeared after factor graph conversion.
- Note the factor graph conversion can be directly from a DGM.

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

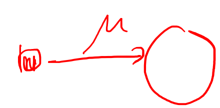
Factor Graphs: Sum-Product Algorithm

- **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:

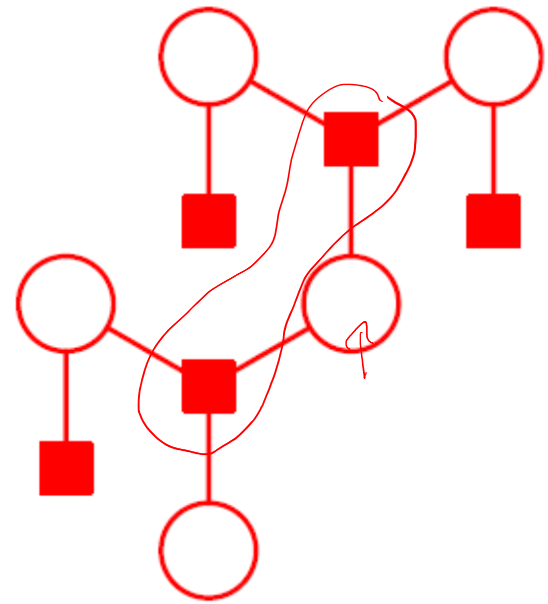


1. Messages υ: flow from variable to factor nodes.
2. Messages μ: flow from factor to variable nodes.



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.



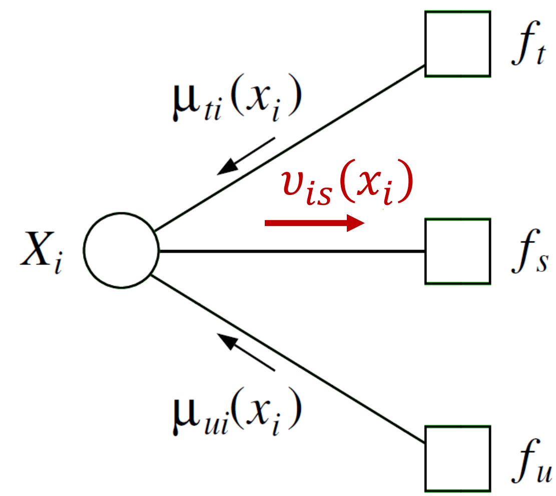
Messages from Variable to Factor Nodes

var. i

s factor node

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

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



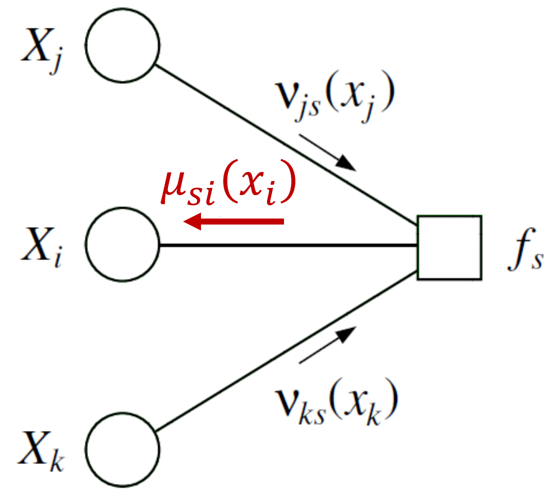
- The product is taken over all incoming messages to the variable node X_i, other than the factor node f_s.

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 :

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



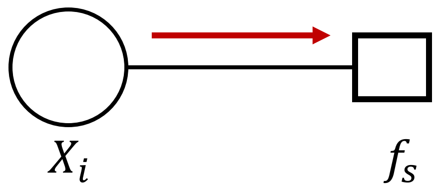
- The product is taken over all incoming messages to the factor node f_s, other than the variable node X_i.

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

Messages From The Leaf Nodes

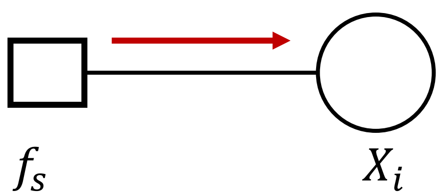
- Message from a leaf variable node to factor node:

$$\nu_{is}(x_i) = 1$$



- 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.

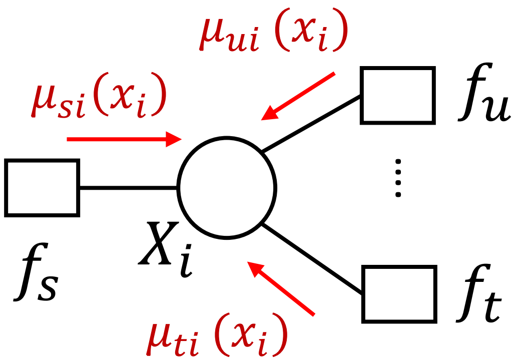
Applies to both variable and factor nodes.

Marginal Probability of a Node

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

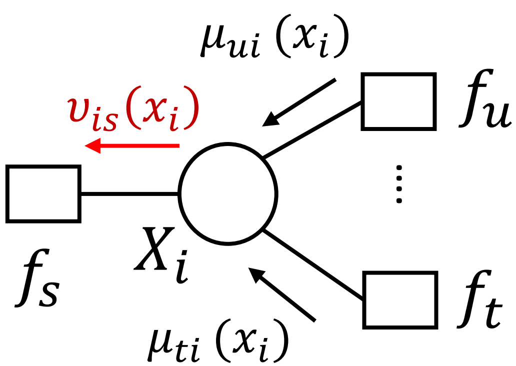
$$p(x_i) \propto \prod_{s \in \mathcal{N}(i)} \mu_{si}(x_i)$$

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



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. $\textsc{Evidence}(E)$
   $f = \textsc{ChooseRoot}(\mathcal{V})$
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 $j \in \mathcal{N}(s) \backslash i$
   $\nu\text{-}\textsc{Collect}(s, j)$
   $\mu\text{-}\textsc{SendMessage}(s, i)$

run on factor nodes

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

run on variable nodes

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

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

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

1. $\text{EVIDENCE}(E)$
   $f = \text{CHOOSEROOT}(\mathcal{V})$
2. **for** $s \in \mathcal{N}(f)$
   $\mu\text{-COLLECT}(f, s)$
3. **for** $s \in \mathcal{N}(f)$
   $\nu\text{-DISTRIBUTE}(f, s)$
4. **for** $i \in \mathcal{V}$
   $\text{COMPUTEMARGINAL}(i)$

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

Message from variable node X_i to the factor node f_s:

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

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

$\mu\text{-DISTRIBUTE}(s, i)$
   $\mu\text{-SENDMESSAGE}(s, i)$
   for $t \in \mathcal{N}(i)\backslash s$
   $\nu\text{-DISTRIBUTE}(i, t)$

4. $\text{COMPUTEMARGINAL}(i)$ // 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

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

1. EVIDENCE(E)
   f = CHOOSEROOT($\mathcal{V}$)
2. **for** $s \in \mathcal{N}(f)$
   μ-COLLECT(f, s)
3. **for** $s \in \mathcal{N}(f)$
   ν-DISTRIBUTE(f, s)
4. **for** $i \in \mathcal{V}$
   COMPUTEMARGINAL(i)

3. ν-DISTRIBUTE(i, s) // distribute messages from root to leaves
   ν-SENDMESSAGE(i, s)
   for $j \in \mathcal{N}(s) \backslash i$
   μ-DISTRIBUTE(s, j)

   μ-DISTRIBUTE(s, i)
   μ-SENDMESSAGE(s, i)
   for $t \in \mathcal{N}(i) \backslash s$
   ν-DISTRIBUTE(i, t)

Message from variable node X_i to the factor node f_s:

ν-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:

μ-SENDMESSAGE(s, i)

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

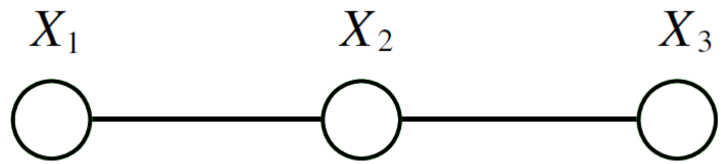
4. COMPUTEMARGINAL(i) // compute **marginal probability**
   $p(x_i) \propto \nu_{is}(x_i) \mu_{si}(x_i)$

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

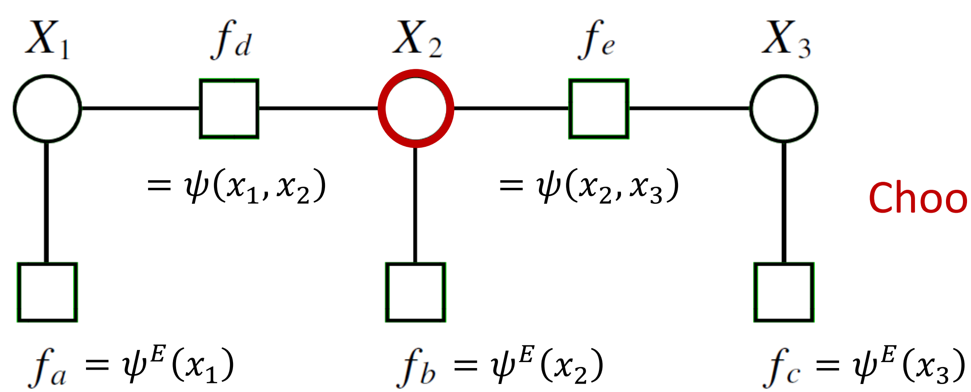
Factor Tree Sum-Product Algorithm

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

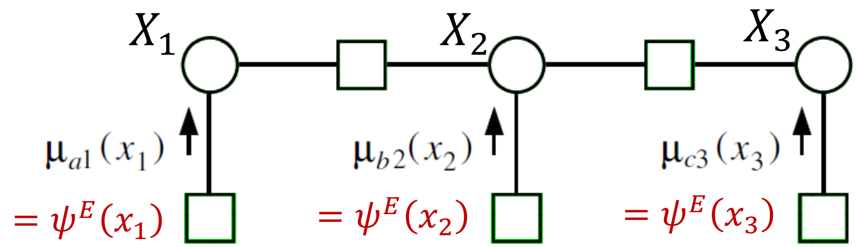


Choose X_2 as root node

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

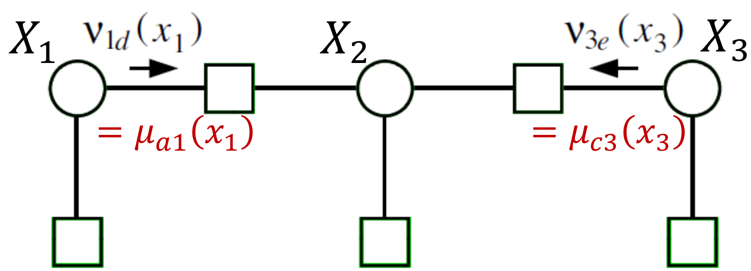
Factor Tree Sum-Product Algorithm

Example:



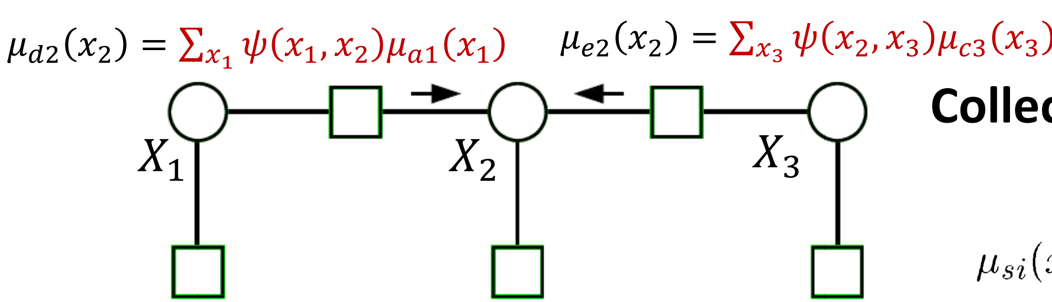
Collect messages from leaf nodes:

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



Collect variable to factor messages:

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



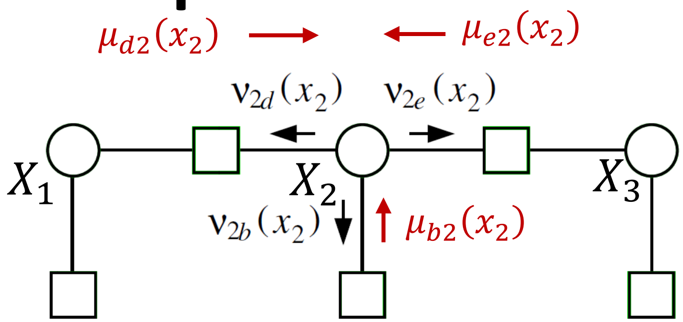
Collect factor to variable messages:

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

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

Factor Tree Sum-Product Algorithm

Example:



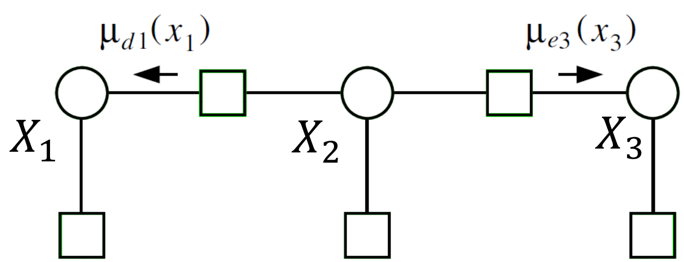
Distribute variable to factor messages:

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

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

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

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

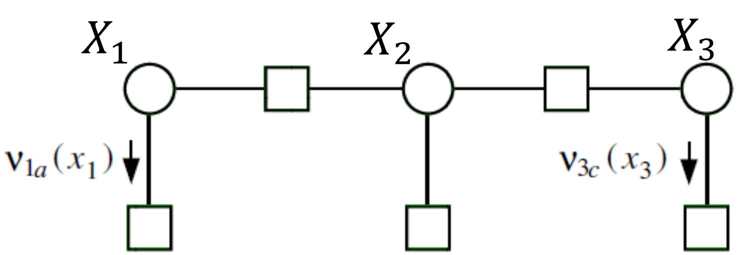


Distribute factor to variable messages:

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

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

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



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.

Exercise: 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 Sketch:

UGM:

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

Factor Graph:

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

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

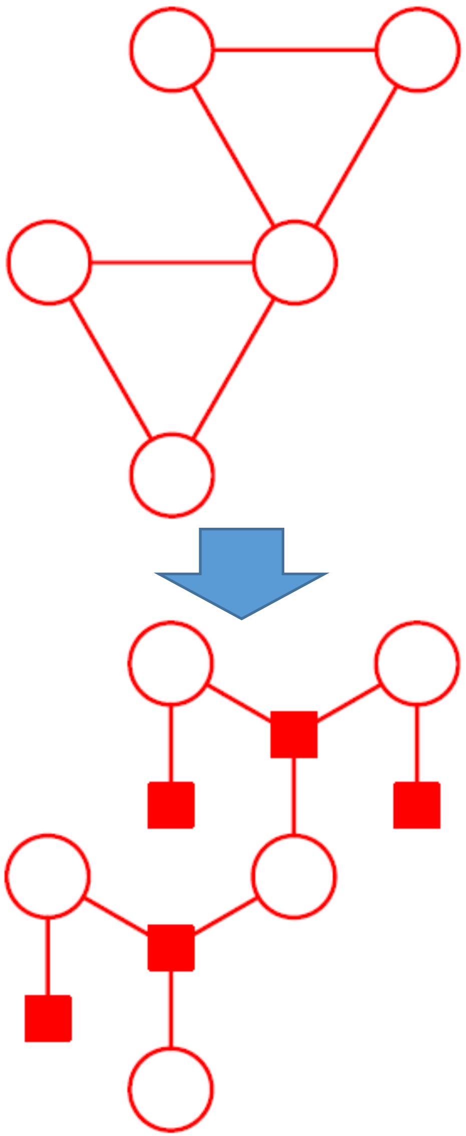
NUS National University of Singapore | School of Computing

Junction Trees

Cluster Graph, Family Preservation, Running-Intersection Property

Factor Graph Idea

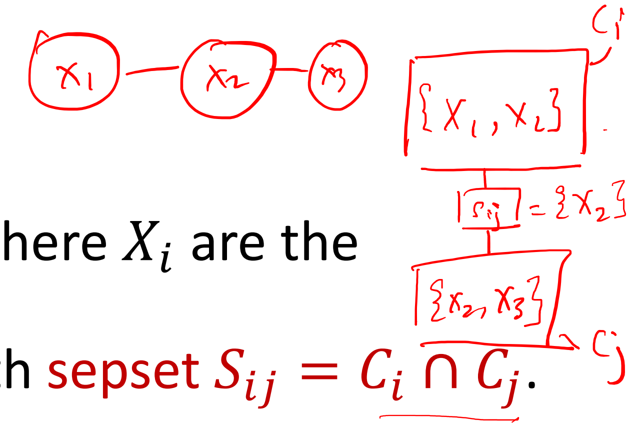
- **Insight:**
  - Convert graph into a Factor tree.
  - Run Belief-propagation on the tree (efficient!)
- Works for PolyTrees, but not in general
- For general graphs, we will use another data structure called a **Junction Tree.**



Junction Tree Algorithm

- Main idea behind Junction Tree Algorithm:
  - 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 & Family Preservation



- Undirected graph such that:
  1. Nodes are clusters $C_i \subseteq \{X_1, \ldots, X_n\}$, 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 $\Psi \in \{\psi_1, \ldots, \psi_k\}$ from an UGM, we assign each ψ_k to a cluster $C_{\alpha(k)}$ s.t. $\text{Scope}[\psi_k] \subseteq C_{\alpha(k)}$.

$\alpha(k)$

$C_i = \{X_1, X_2, X_3\}$

$\psi_1(X_1, X_2]$

$\text{Scope}[\psi_1] = \{X_1, X_2\}. \subseteq \underbrace{C_{\alpha(k)}}_{=i}$

- Cluster potential is defined as:

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

$\{X_1, X_2\} \subseteq \{X_1, X_2, X_3\}.$ ✓

Cluster Graphs

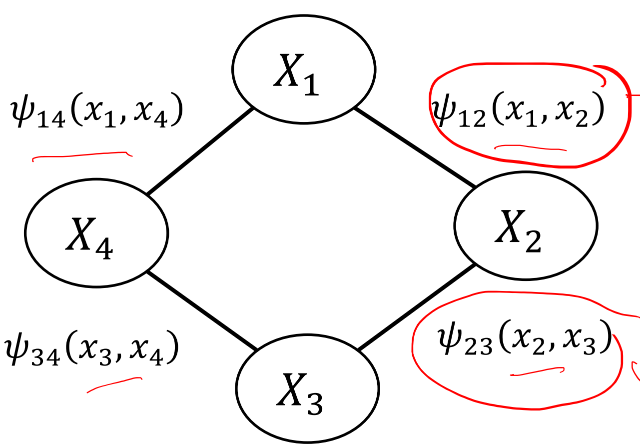
Example:

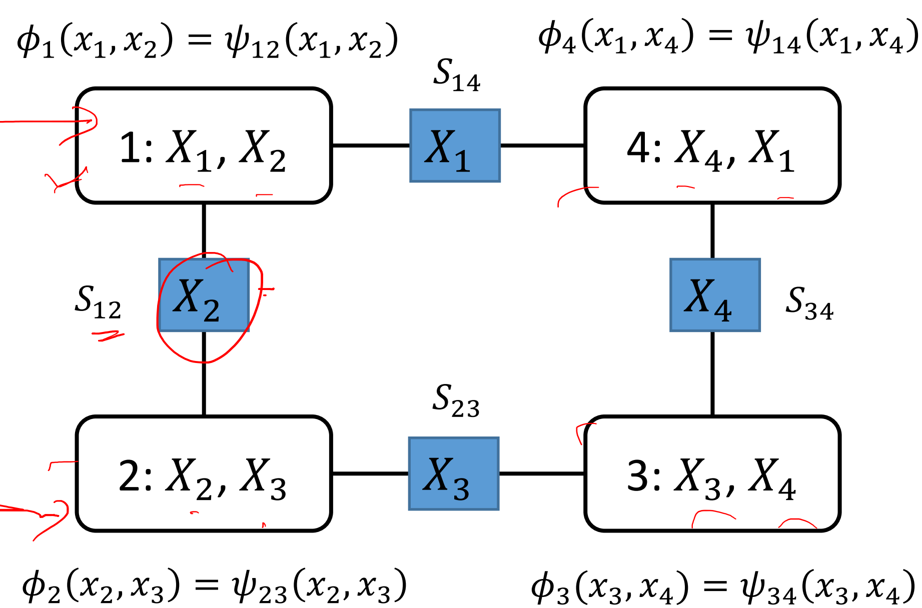
Cluster Graph

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

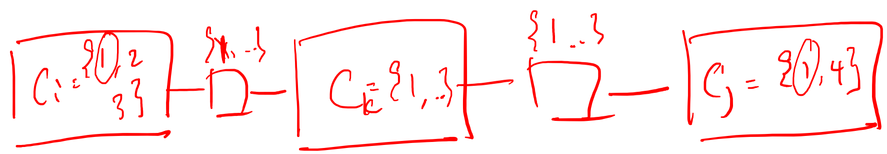




Adapted from: "Probabilistic Graphical Models", Daphne Koller

Running Intersection Property: Junction Tree Property

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



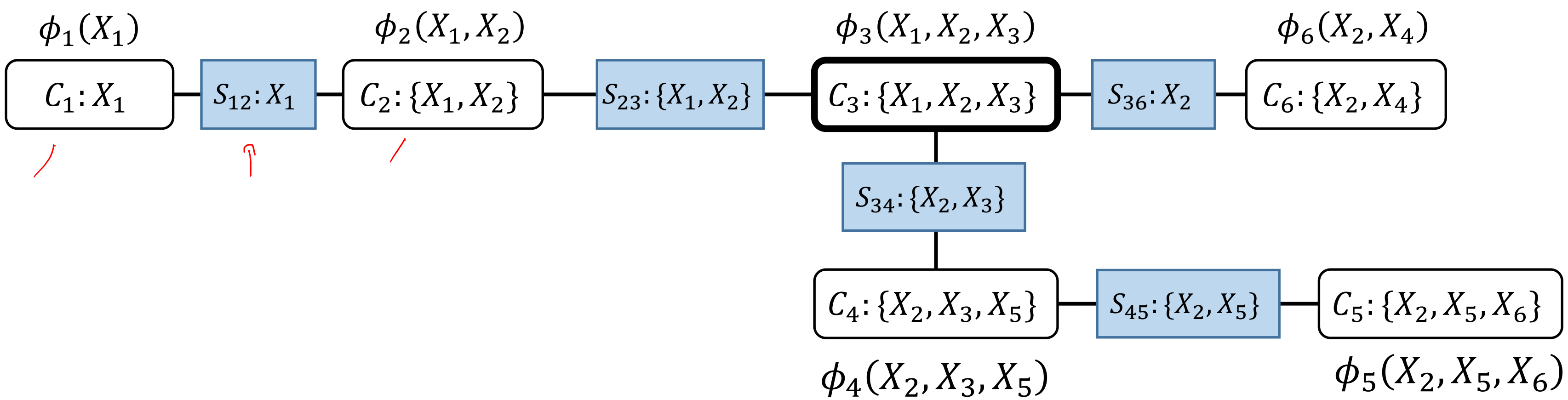
  There exists an unique path between C_i and C_j for which all clusters and sepsets contain X.

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

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".

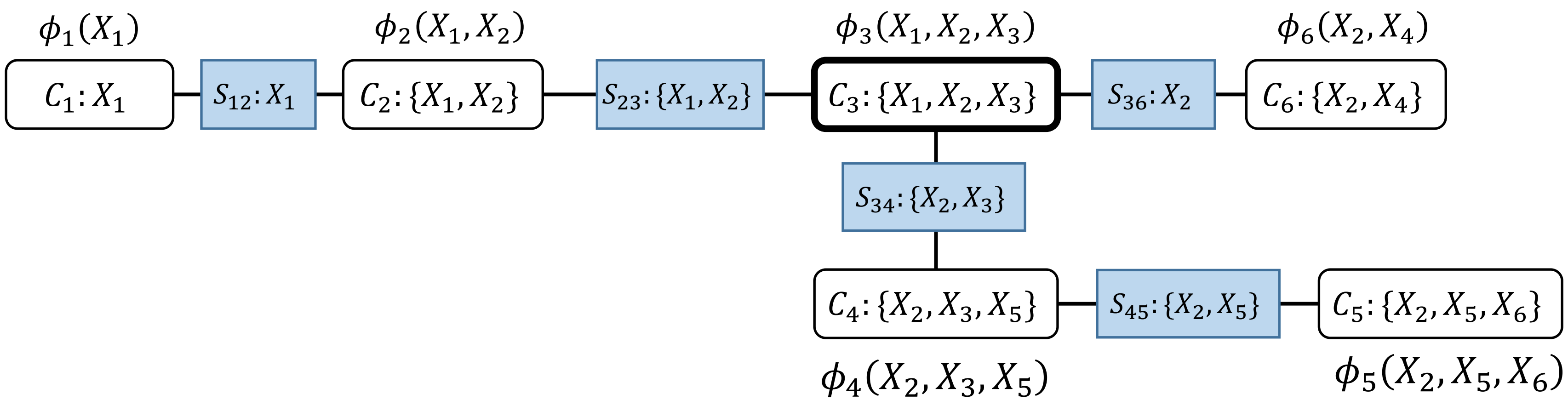
Junction Tree Example



Is this a valid clique tree?

Verify that family preservation and the running intersection property hold

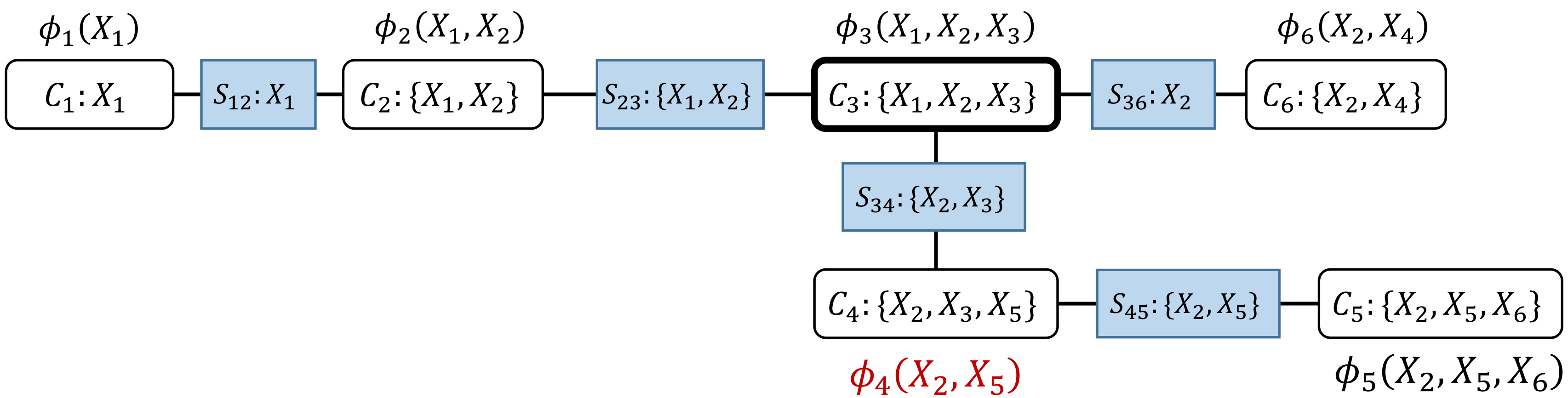
Junction Tree Example



Is this a valid clique tree?

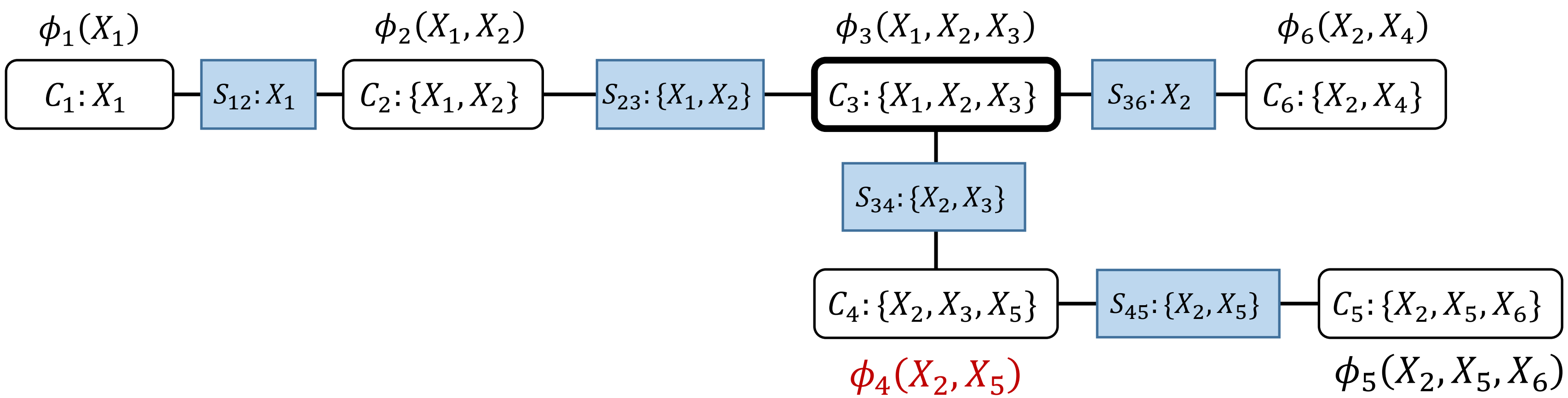
Yes!

Junction Tree Example



Is this a valid clique tree?

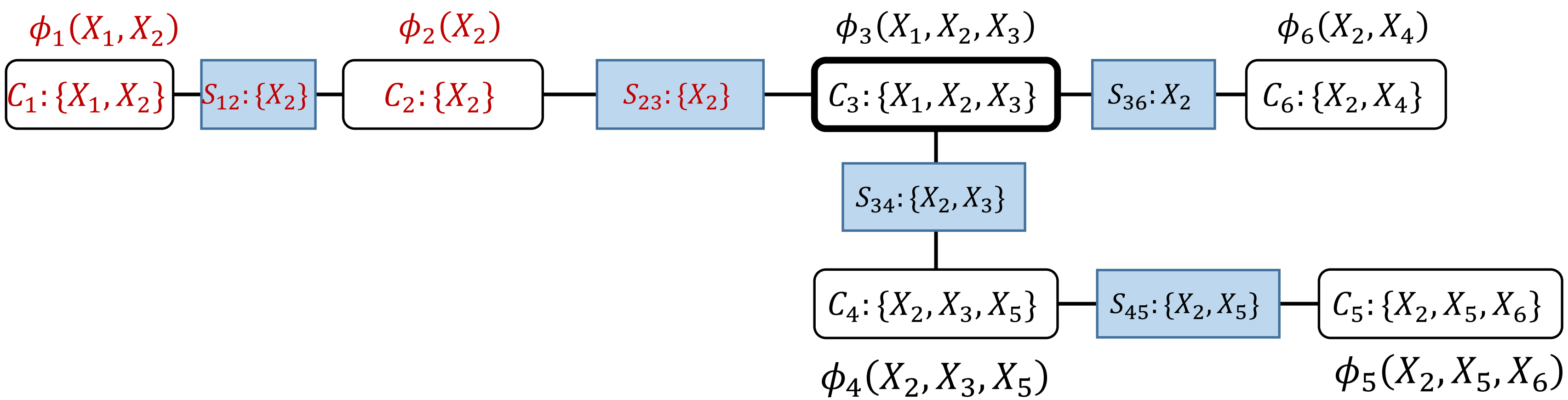
Junction Tree Example



Is this a valid clique tree?

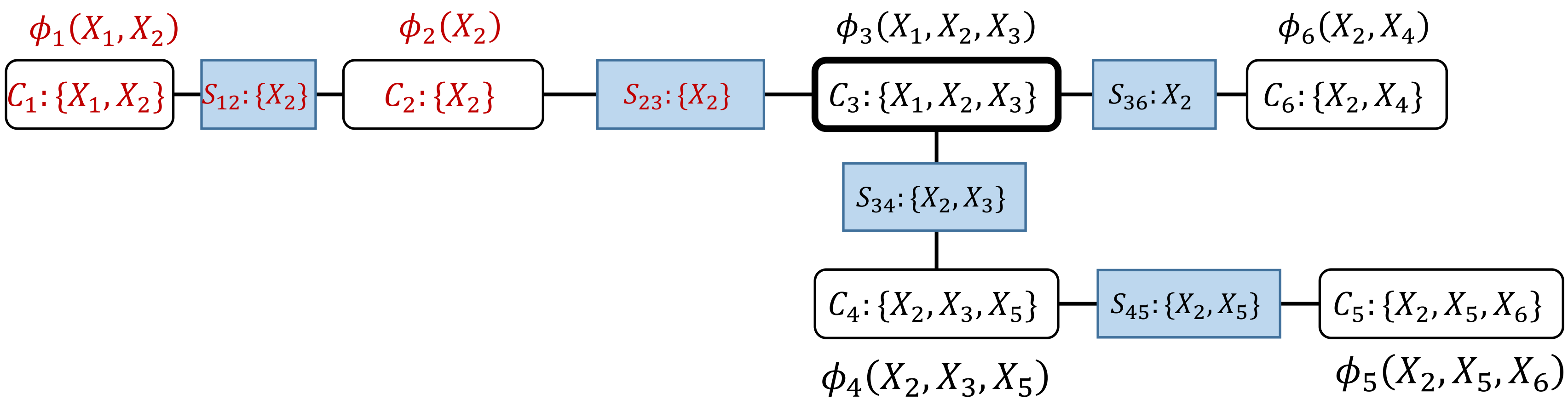
Yes!

Junction Tree Example



Is this a valid clique tree?

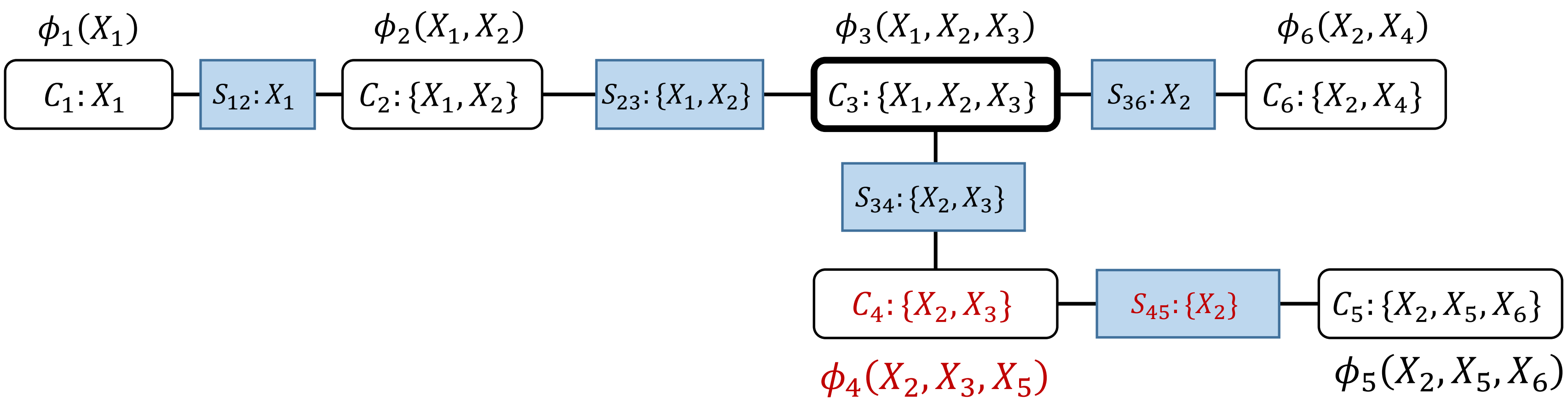
Junction Tree Example



Is this a valid clique tree?

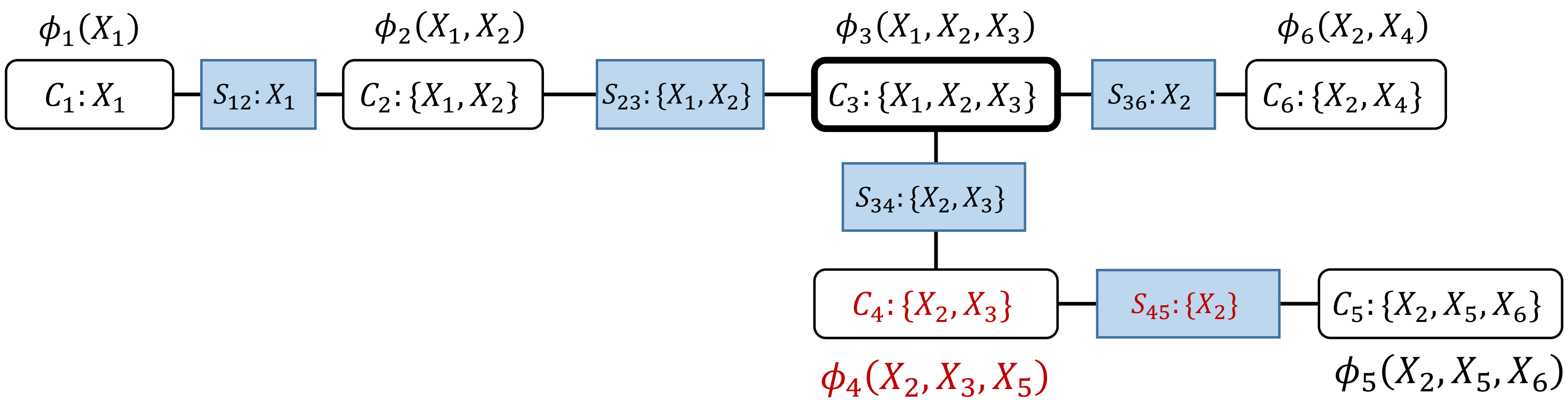
No! Running intersection fail.

Junction Tree Example



Is this a valid clique tree?

Junction Tree Example



Is this a valid clique tree?

No! Family Preservation Fail.

NUS National University of Singapore | School of Computing

Sum-Product on Junction Trees

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.

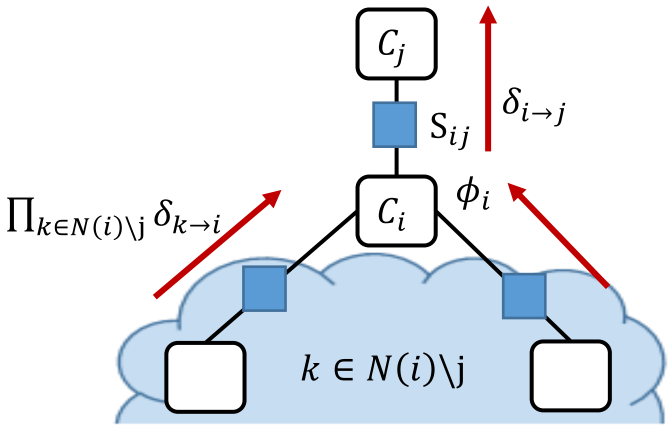
Junction Tree : Sum-Product Algorithm

- 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

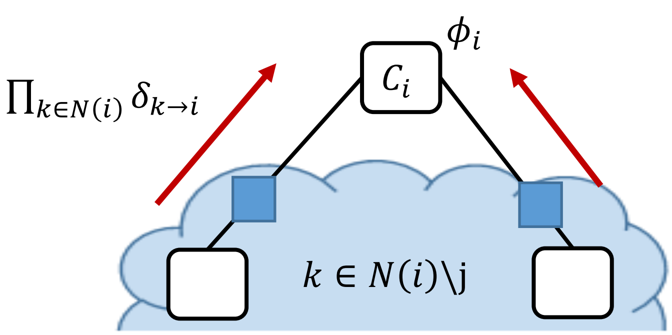
- Use the sum-product algorithm to compute messages from C_i to C_j:

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



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

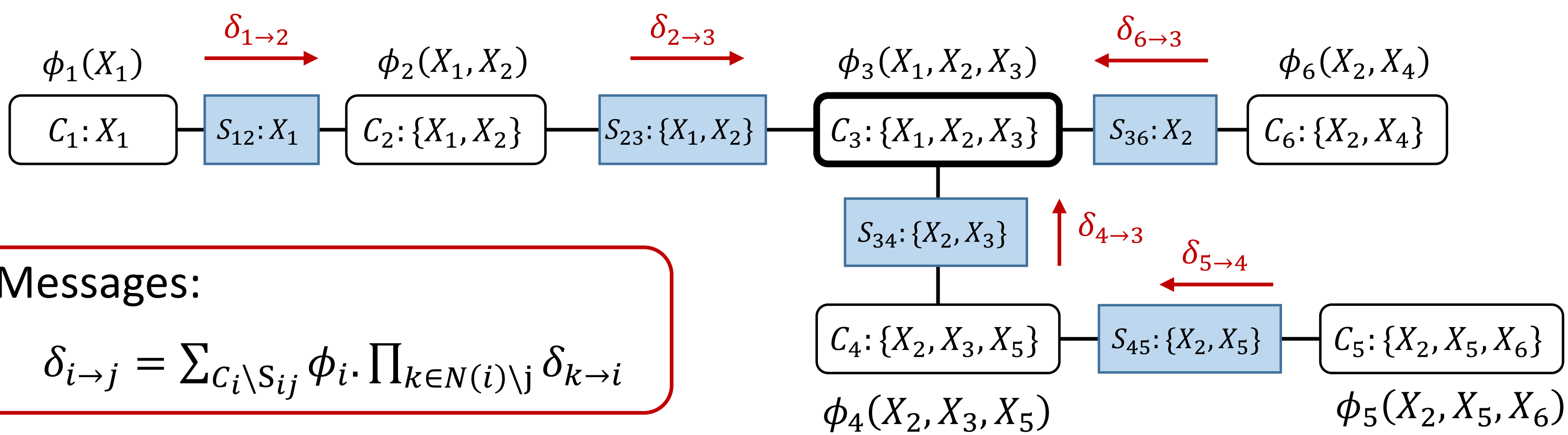
$$\tilde{p}(C_i) = \phi_i . \prod_{k\in N(i)} \delta_{k\to i}$$



*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:

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

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

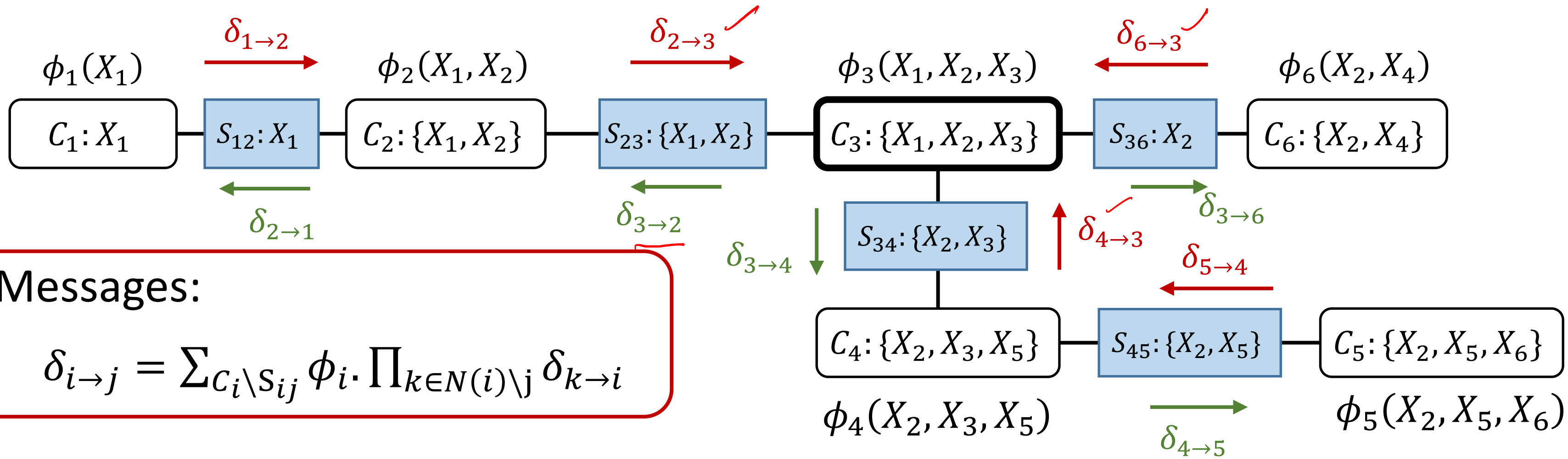
$$\delta_{5\to4} = \sum_{C_5\setminus S_{45}} \phi_5 = \sum_{X_6} \phi_5$$

$$\delta_{4\to3} = \sum_{C_4\setminus S_{34}} \phi_4 \,.\, \delta_{5\to4} = \sum_{X_5} \phi_4 \sum_{X_6} \phi_5$$

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

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:

$$\delta_{1\to 2} = \sum_{C_1 \setminus S_{12}} \phi_1 = \phi_1$$

$$\delta_{2\to 3} = \sum_{C_2 \setminus S_{23}} \phi_2 \,.\, \delta_{1\to 2} = \phi_2 . \phi_1$$

$$\delta_{5\to 4} = \sum_{C_5 \setminus S_{45}} \phi_5 = \sum_{X_6} \phi_5$$

$$\delta_{4\to 3} = \sum_{C_4 \setminus S_{34}} \phi_4 \,.\, \delta_{5\to 4} = \sum_{X_5} \phi_4 \sum_{X_6} \phi_5$$

$$\delta_{6\to 3} = \sum_{C_6 \setminus S_{36}} \phi_6 = \sum_{X_4} \phi_6$$

Outward pass:

$$\delta_{3\to 2} = \sum_{C_3 \setminus S_{23}} \phi_3 \,.\, (\delta_{6\to 3} . \delta_{4\to 3}) = \sum_{X_3} \phi_3 \, \delta_{63} \, \delta_{43}$$

$$\delta_{2\to 1} = \sum_{C_2 \setminus S_{12}} \phi_2 \,.\, \delta_{3\to 2}$$

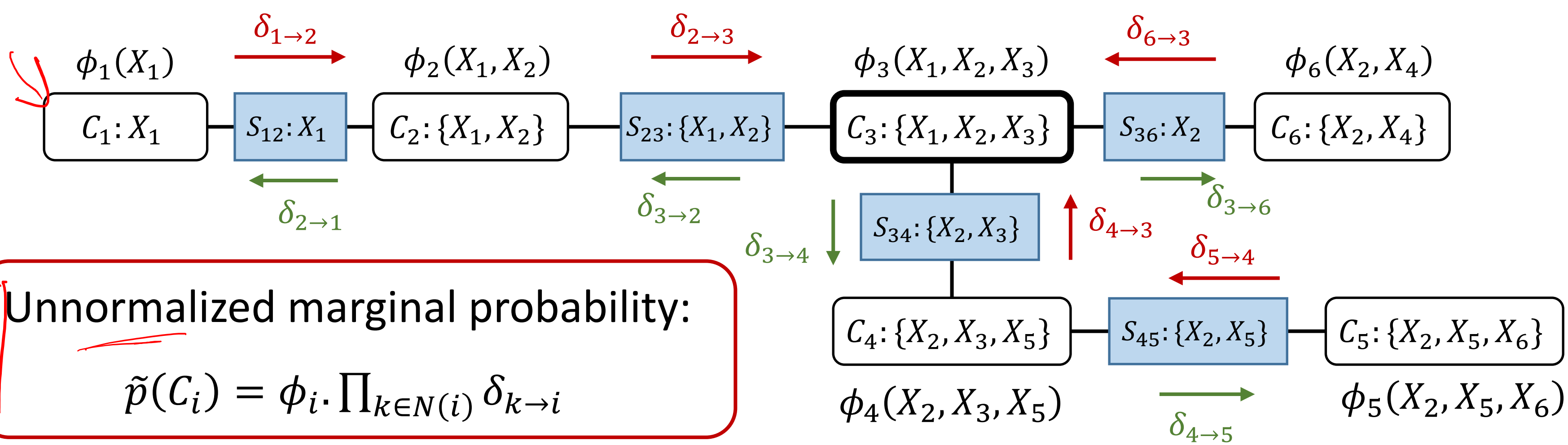
$$\delta_{3\to 6} = \sum_{C_3 \setminus S_{36}} \phi_3 \,.\, \delta_{2\to 3} \,.\, \delta_{4\to 3}$$

$$\delta_{3\to 4} = \sum_{C_3 \setminus S_{34}} \phi_3 \,.\, \delta_{2\to 3} \,.\, \delta_{6\to 3}$$

$$\delta_{4\to 5} = \sum_{C_4 \setminus S_{45}} \phi_4 \,.\, \delta_{3\to 4}$$

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Unnormalized marginal probability:

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

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

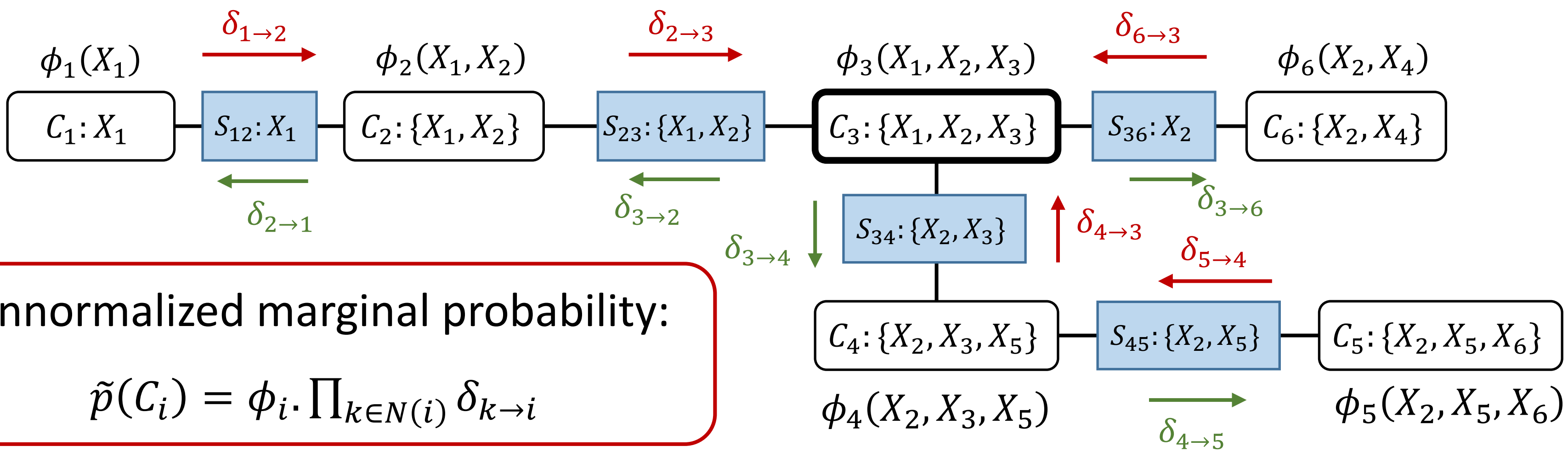
Marginal probability:

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

Result is equivalent to variable elimination!

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Unnormalized marginal probability:

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

$$\tilde{p}(C_2) = \tilde{p}(X_1, X_2)$$
$$= \phi_2 . \prod_{k \in N(2)} \delta_{k\to 2}$$
$$= \phi_2 . \delta_{1\to2} . \delta_{3\to2}$$

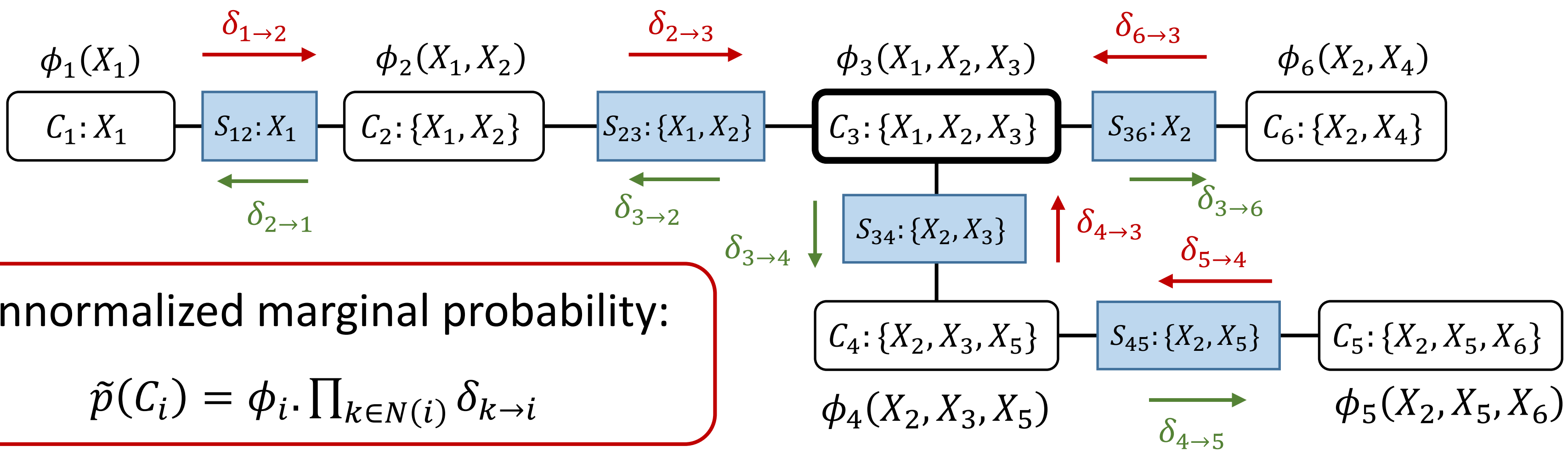
Marginal probabilities:

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

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

Junction Tree : Sum-Product Algorithm

Example: Let's choose C_3 as the root



Unnormalized marginal probability:

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

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

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

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

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

NUS National University of Singapore | School of Computing

Constructing the Junction Tree

Constructing the Junction Tree

1. Triangulation via graph elimination
2. Obtain clusters (cliques generated via elimination) and all possible sepsets
3. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.

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

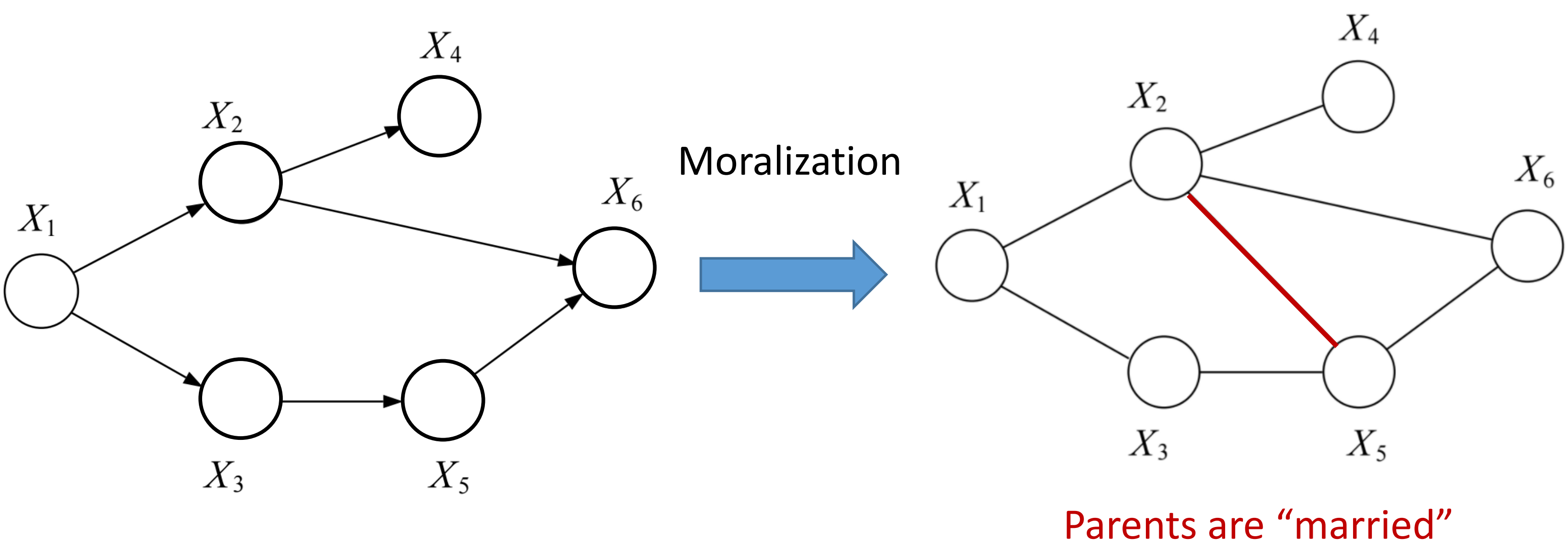
    for each node X_i in I
        connect all of the remaining neighbors of X_i
        remove X_i from the graph
    end

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 $I = (6; 5; 4; 3; 2; 1)$

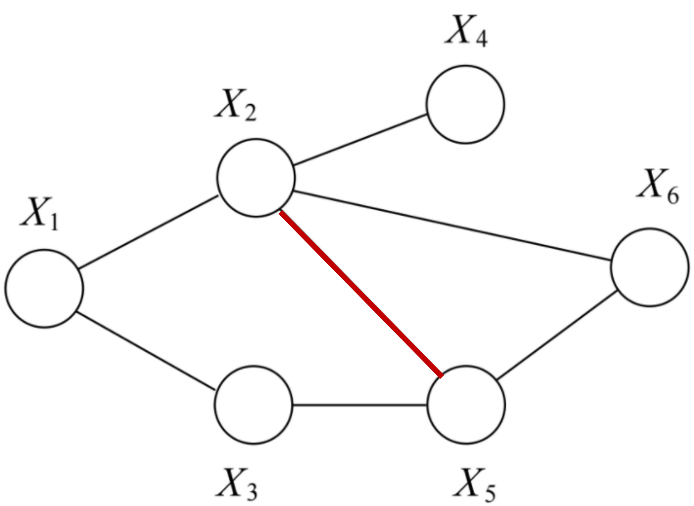


Source: "An introduction to probabilistic graphical models", Michael I. Jordan, 2002.

Constructing the Junction Tree

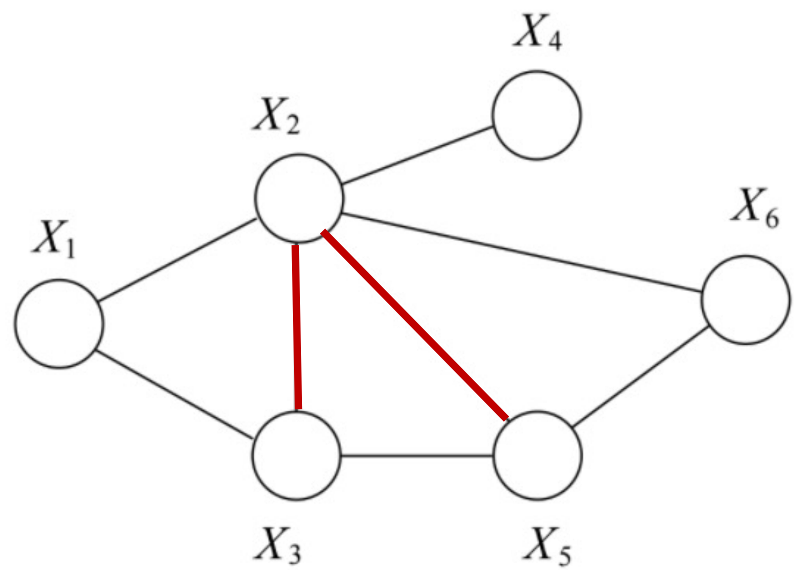
1. Triangulation: Get the reconstituted graph

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



Parents are "married"

Undirected Graph Eliminate

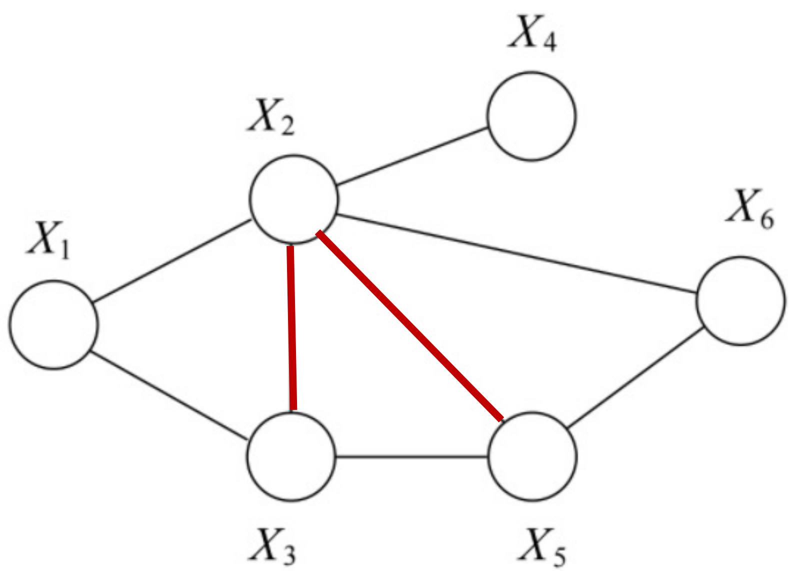


Reconstituted graph: additional edges (red) added during the elimination process

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 elimination cliques as clusters, sepset is $S_{ij} = C_i \cap C_j$.



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

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

$C_6: \{X_2, X_4\}$

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

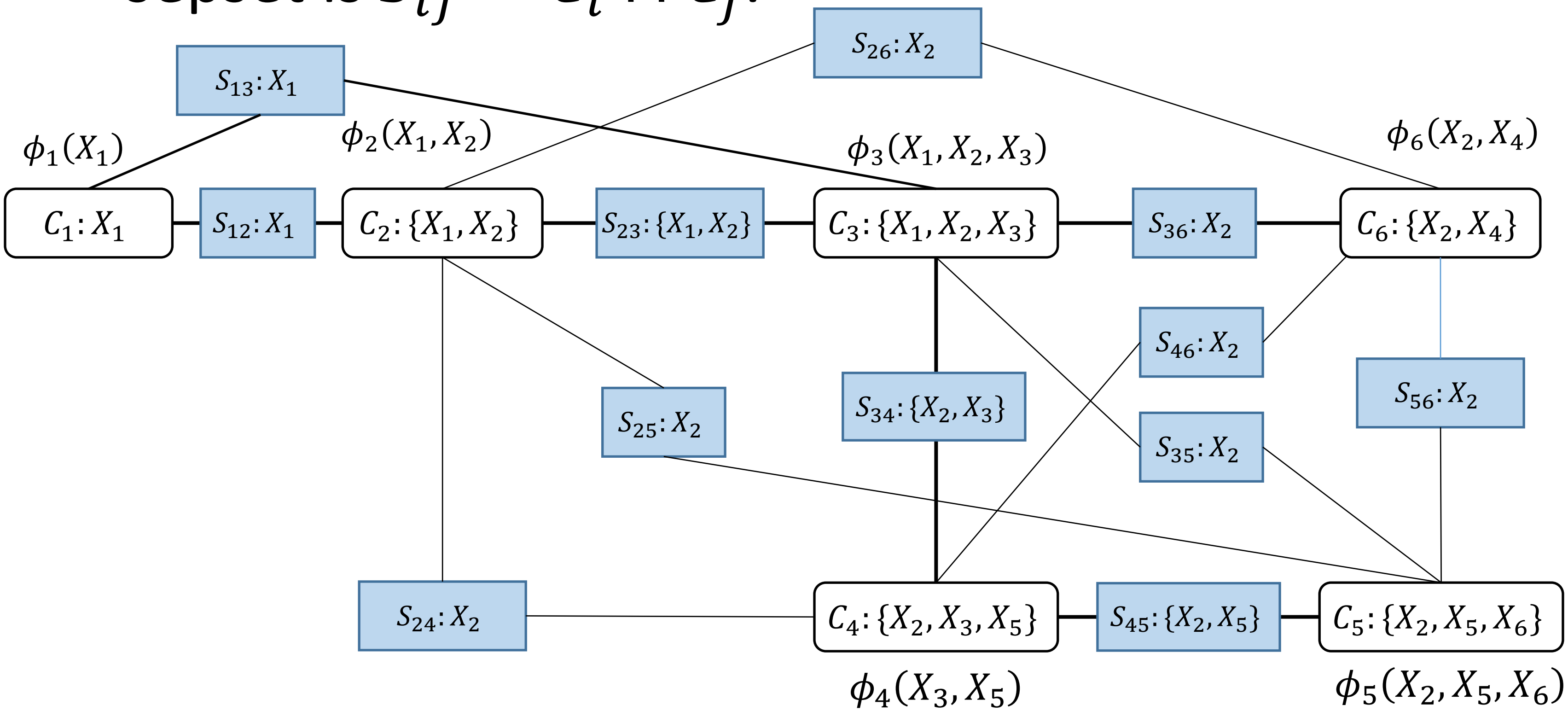
$C_2: \{X_1, X_2\}$

$C_1: X_1$

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

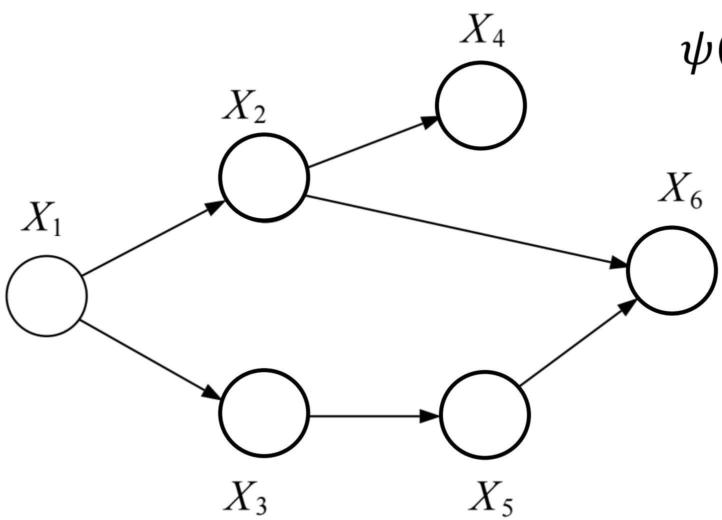
Constructing the Junction Tree

2. Get all clusters and all possible sepsets: Use eliminate cliques as clusters, 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!

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

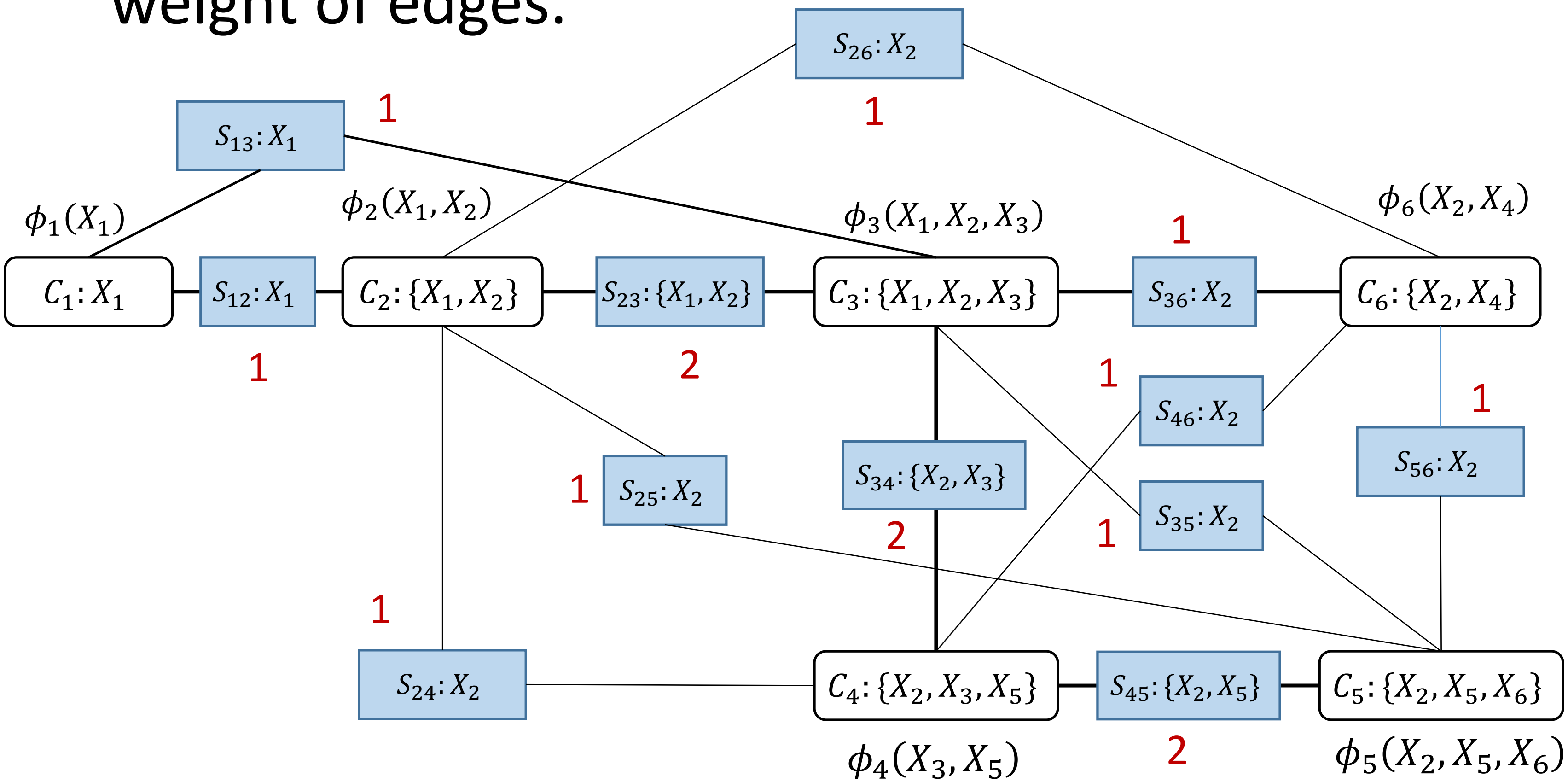
$\phi_3(X_1,X_2,X_3) = p(x_3|x_1)$, $\qquad \phi_4(X_3,X_5) = p(x_5|x_3)$

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

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

Constructing the Junction Tree

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



Constructing the Junction Tree

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

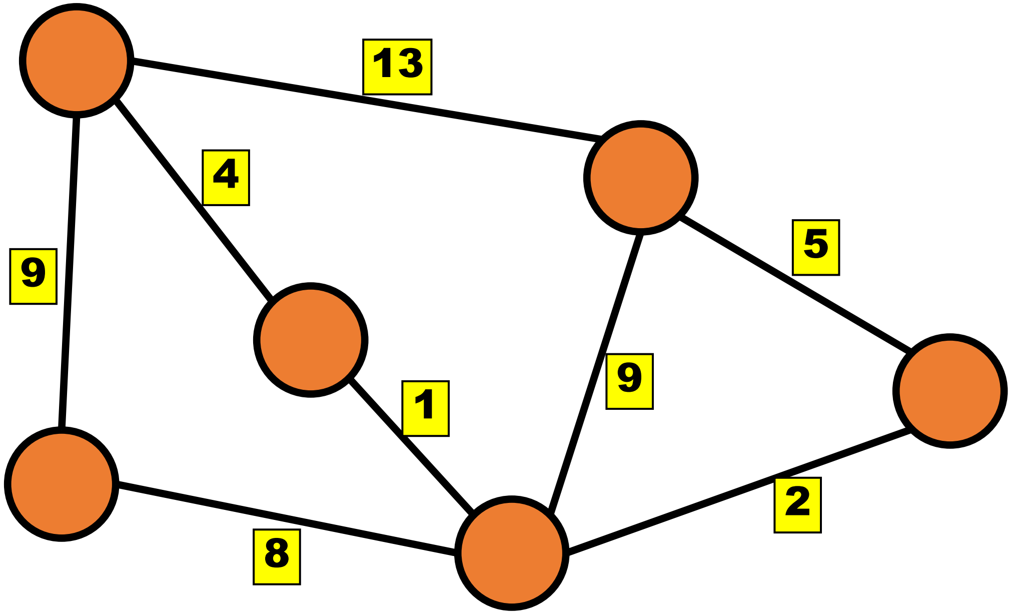
Theorem: A cluster tree T is a clique tree / junction tree only if it is a maximal spanning tree.

Exercise: Prove this.

From CS2040:
Data Structures and Algorithms

- Adapted from my other course.
- Imagine you're a 1st year undergrad again. ☺

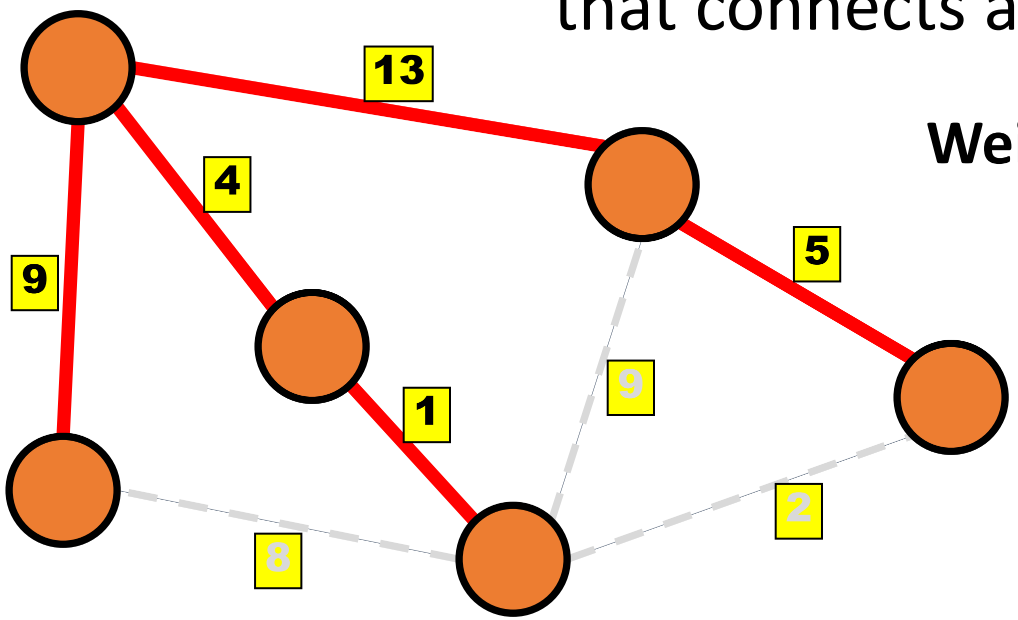
we have the following graph



NUS National University of Singapore | School of Computing

spanning tree: definition

A **spanning tree** is an acyclic subset of the edges that connects all nodes



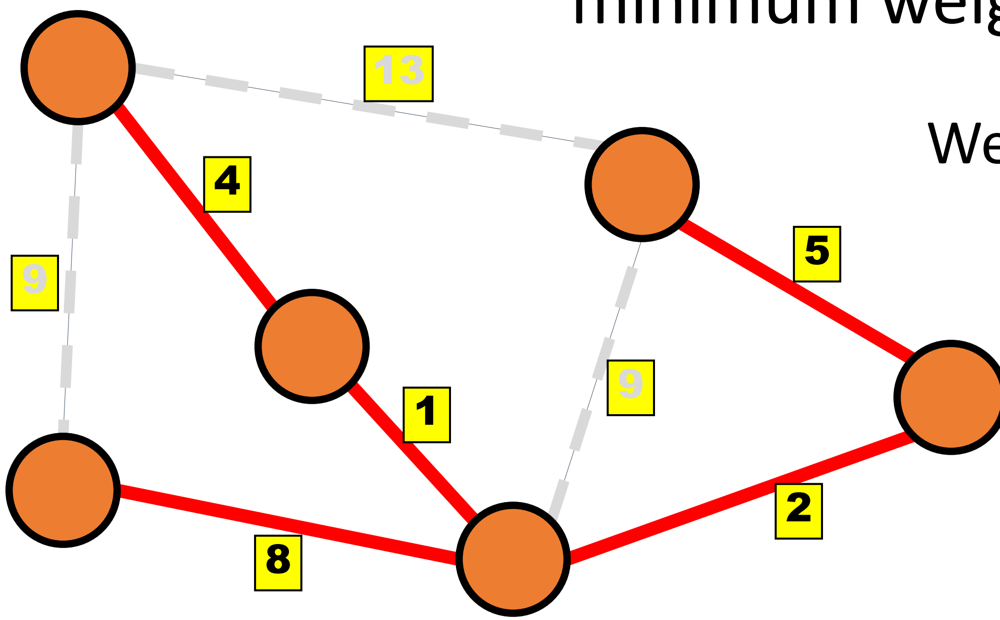
Weight: 32

NUS National University of Singapore | School of Computing

minimum spanning Tree

A spanning tree with minimum weight

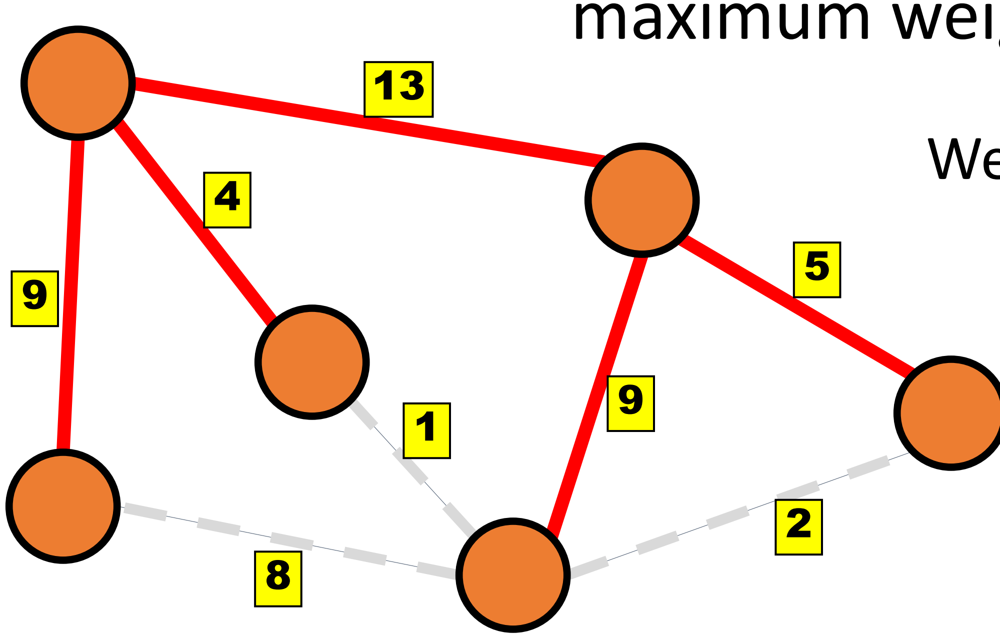
Weight: 20



School of Computing

maximum spanning tree

A spanning tree with maximum weight



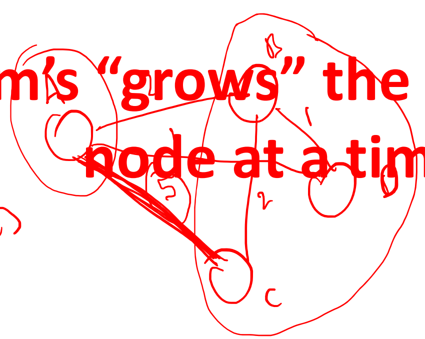
Weight: 40

Prim's algorithm

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V - S\}$
  - Find minimum/maximum weight edge on cut.
  - Add new node to S.

Prim's "grows" the tree one node at a time.



$S = \{A\}$

$S = \{A, C\}$

Analysis:

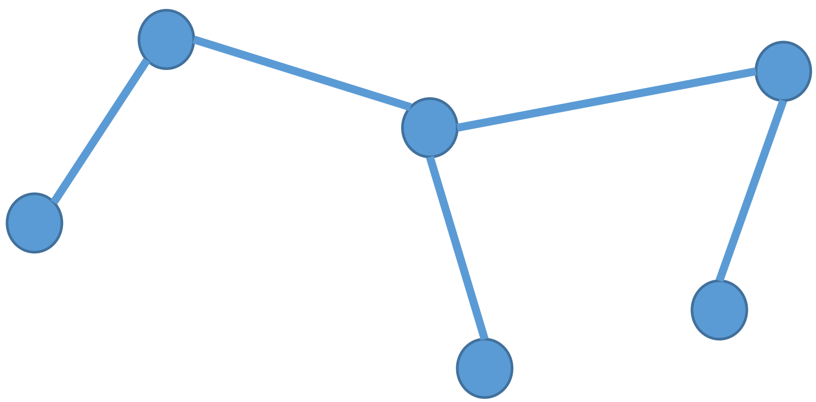
Each vertex added/removed once from the priority queue:

$O(V \log V)$

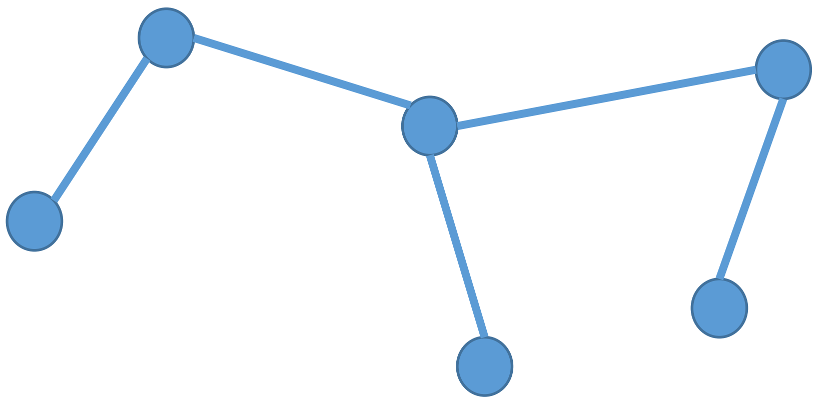
Each edge → one decreaseKey:

$O(E \log V)$

Prim's algorithm



another strategy?

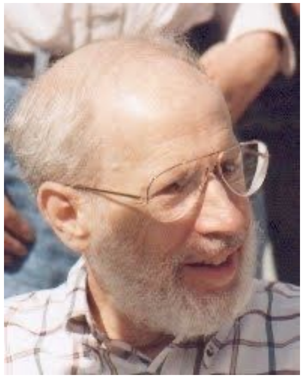


Kruskal's algorithm

(Kruskal 1956)

Basic idea:

- Graph F : a set of trees (initially each vertex is a separate tree)
- Set of edges $S = \{e \in E\}$
- While S is nonempty and F is not spanning:
- Remove minimum/maximum weight edge from S
- If removed edge connects two trees
  - add it to the F (combine the trees)



1925-2010

Kruskal "merges" smaller trees into a bigger tree

Back to CS5340 …

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), decreasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(u, v)
return A
```

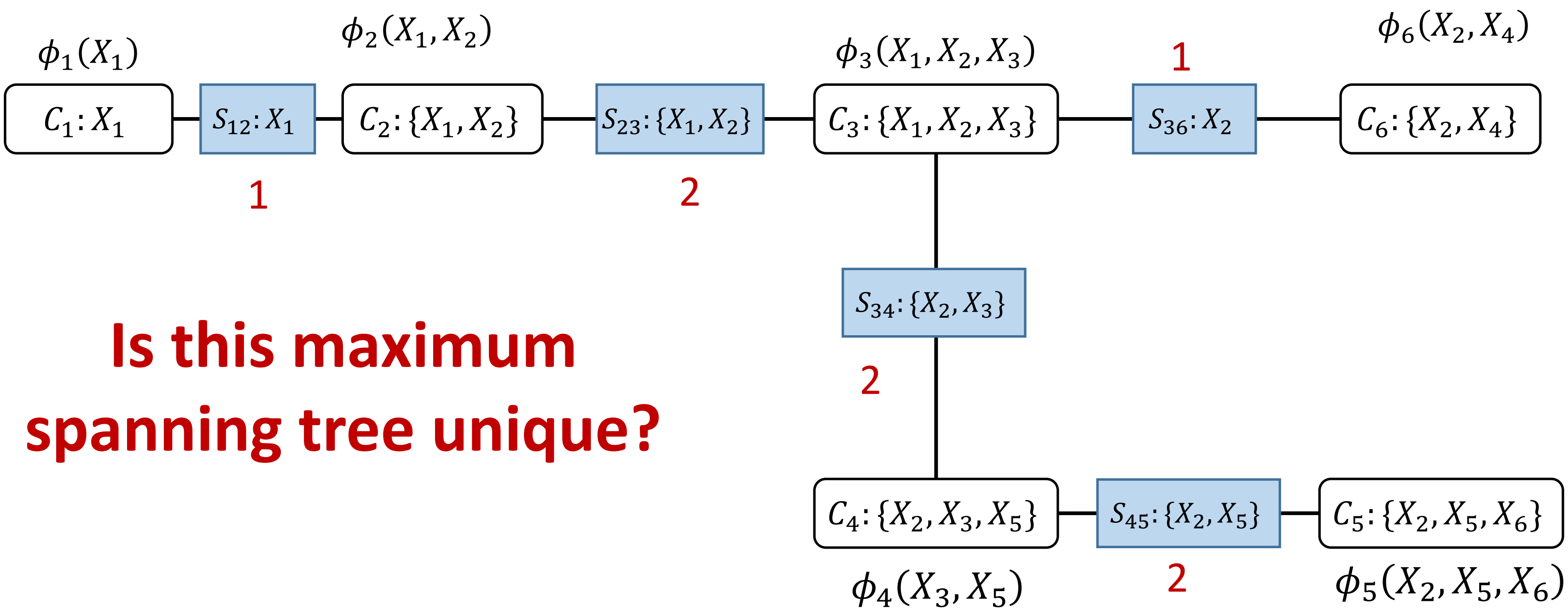
Can be more than 1 maximum spanning tree!

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

Constructing the Junction Tree

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

Example:

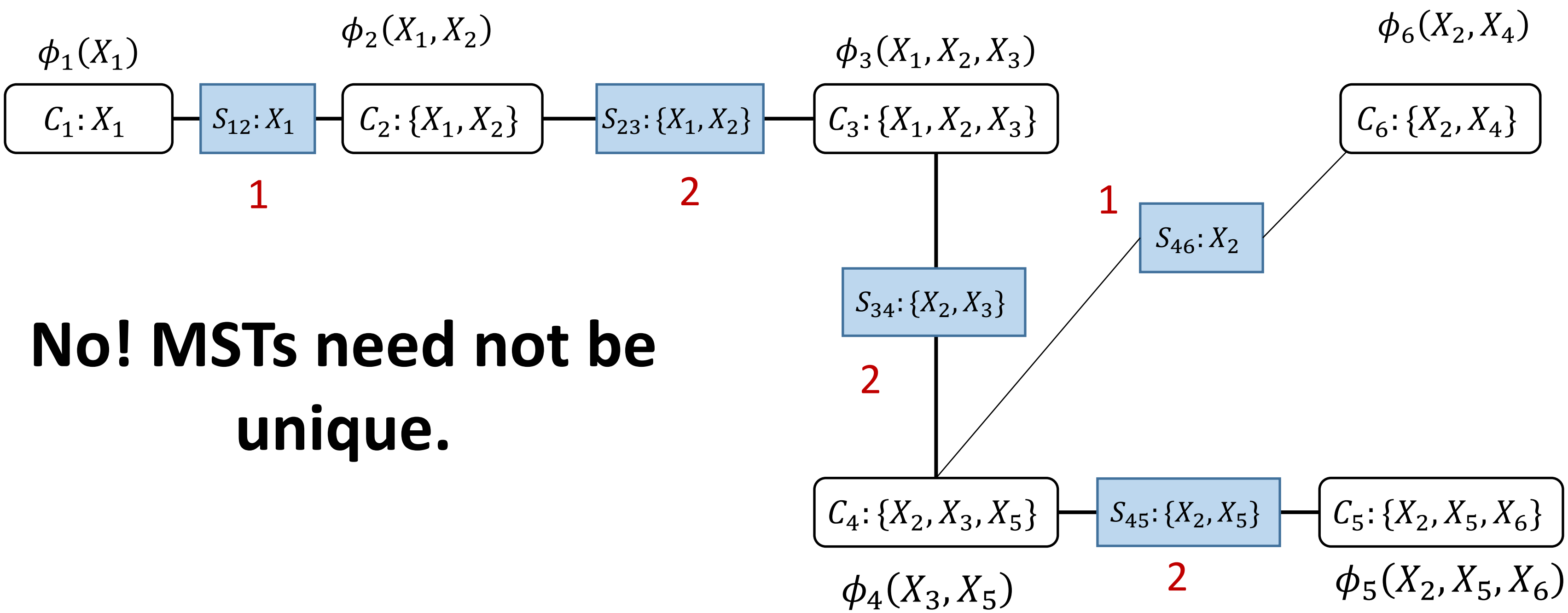


Is this maximum spanning tree unique?

Constructing the Junction Tree

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

Example:

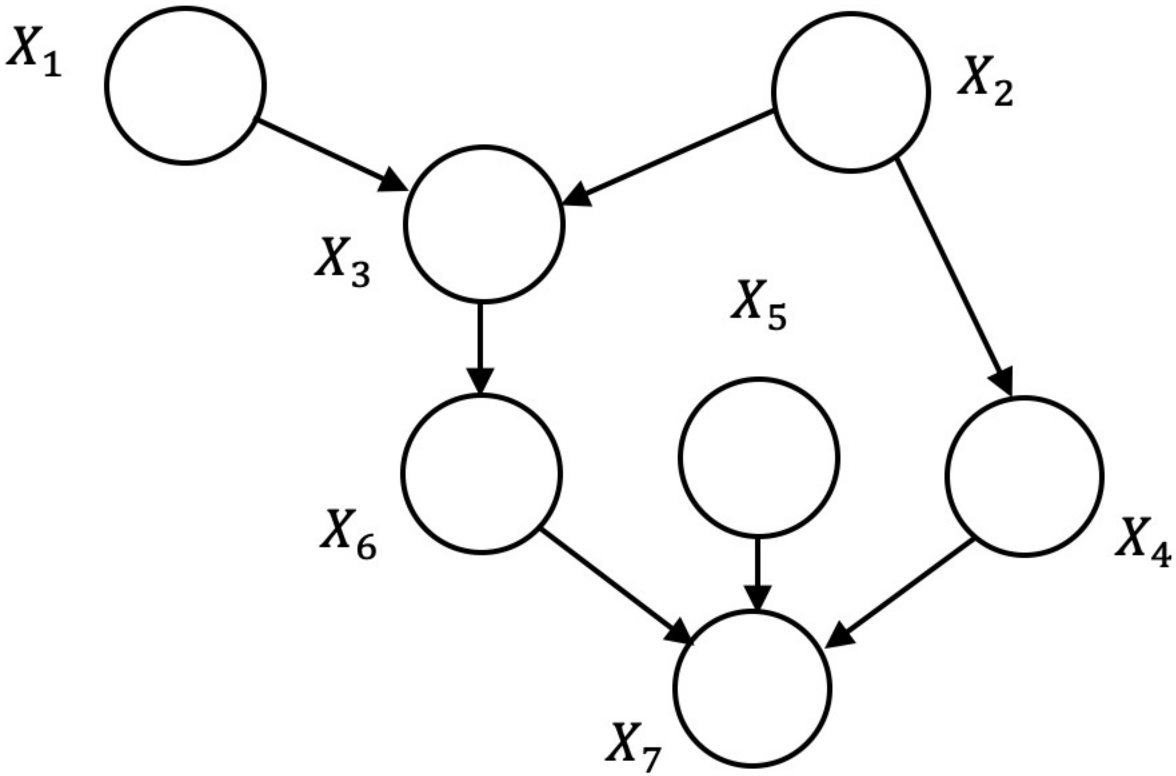


No! MSTs need not be unique.

Constructing the Junction Tree

1. Triangulation via graph elimination
2. Obtain clusters (cliques generated via elimination) and all possible sepsets
3. Assign cluster potentials to the clusters. Respect family preservation.
4. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.

Junction Tree Tutorial



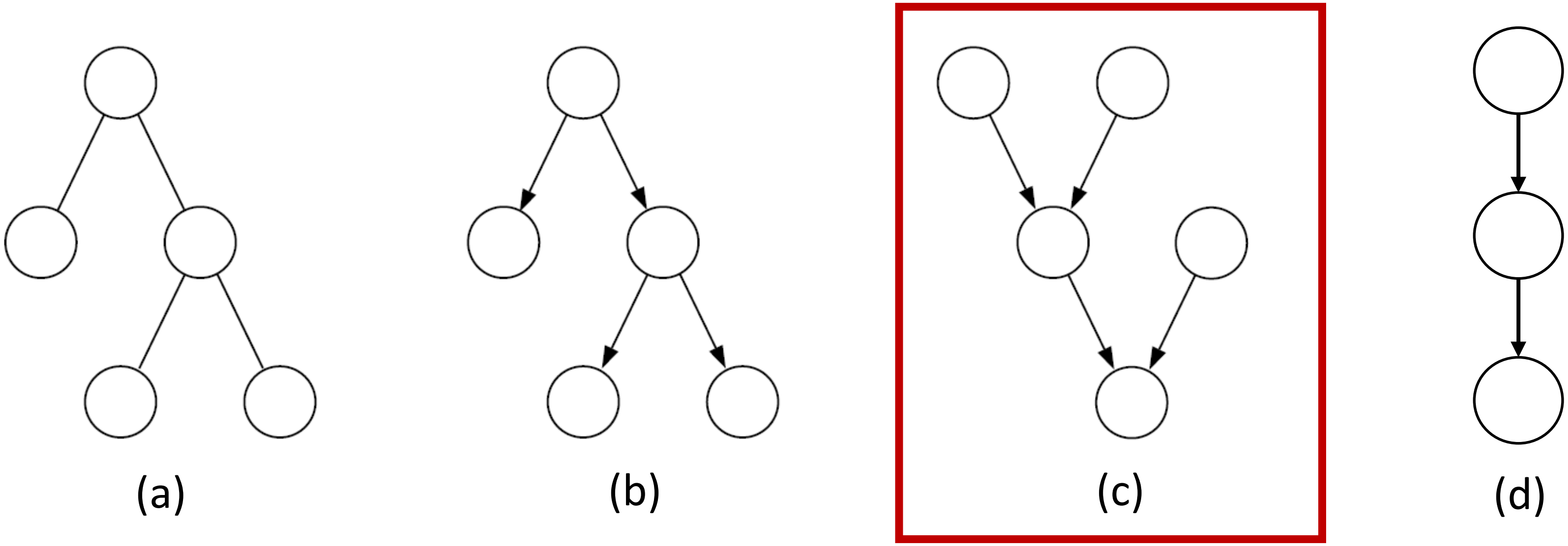
Computational Complexity

- General inference is NP-hard
- Junction tree algorithm does not reduce this complexity.
  - Even with a good ordering, it is possible to construct cases where the cliques are large (e.g., a lattice)
- However, NP-hardness is a worst-case result
- We will learn about approximate algorithms in the coming weeks
  - Note: approximate inference is also NP-hard in general.

Recap

Factor Graphs, Junction Tree Algorithm

"Tree-Like" Graphs



a) Undirected tree: without any loop (unique path between any two nodes)

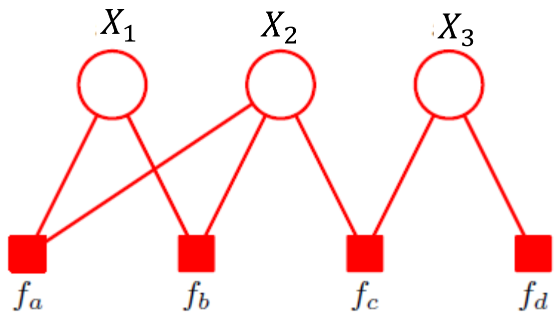
b) Directed tree: only 1 single parent for every node, moralizations lead to an undirected tree.

c) Polytree: nodes with more than 1 parent. Not a directed tree, moralizations lead to loops.

d) Chain: this is also a directed tree (more on chains when we look at Hidden Markov Models).

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

Factor Graphs: Graphical Representation



- A factor graph is a bipartite graph:

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

where

  - vertices $\mathcal{V} \in \{X_1, \ldots, X_n\}$: index the random variables,
  - vertices $\mathcal{F} \in \{\ldots, f_s, \ldots\}$: 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 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 $j \in \mathcal{N}(s)\backslash i$
   $\nu\text{-}\textsc{Collect}(s, j)$
   $\mu\text{-}\textsc{SendMessage}(s, i)$

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

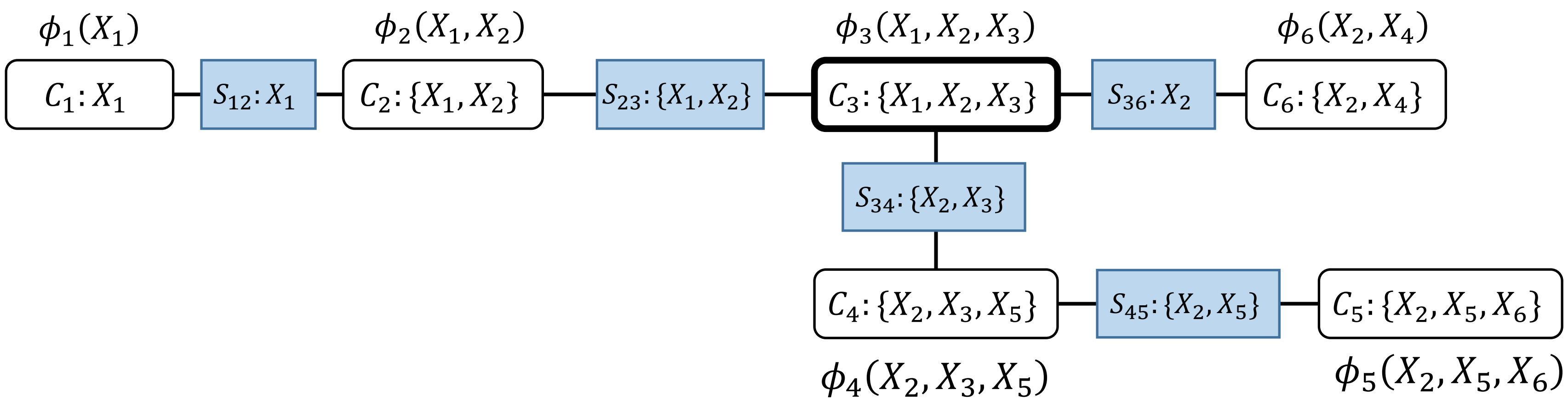
$\nu\text{-}\textsc{Collect}(s, i)$
   for $t \in \mathcal{N}(i)\backslash s$
   $\mu\text{-}\textsc{Collect}(i, t)$
   $\nu\text{-}\textsc{SendMessage}(i, s)$

Message from 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.

Junction Tree Example

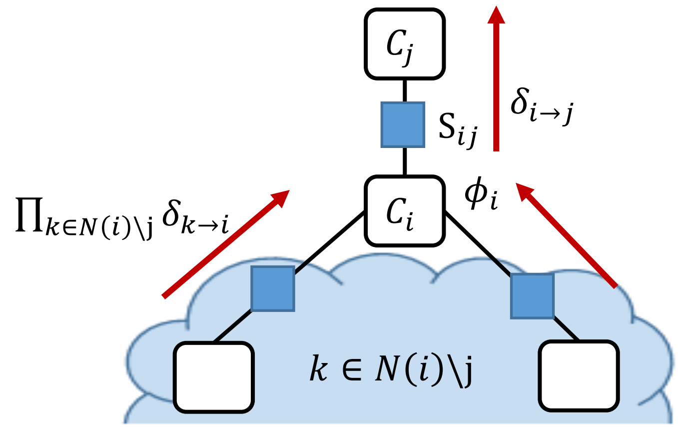


Is this a valid clique tree?

Verify that family preservation and the running intersection property hold

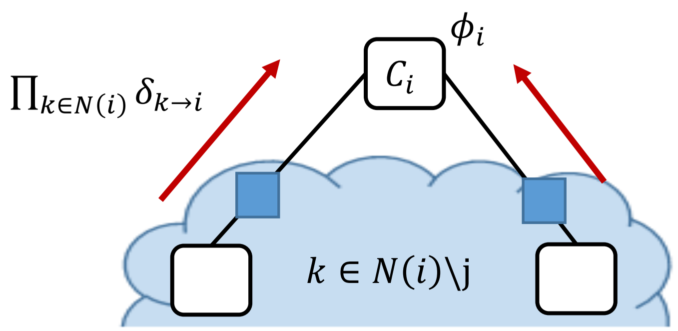
Junction Tree : Sum-Product Algorithm

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



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

- 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

Constructing the Junction Tree

1. Triangulation via graph elimination
2. Obtain clusters (cliques generated via elimination) and all possible sepsets
3. Get clique tree / junction tree: find the maximum spanning tree with cardinality of sepsets as weight of edges.

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