Probabilistic & Unsupervised Learning

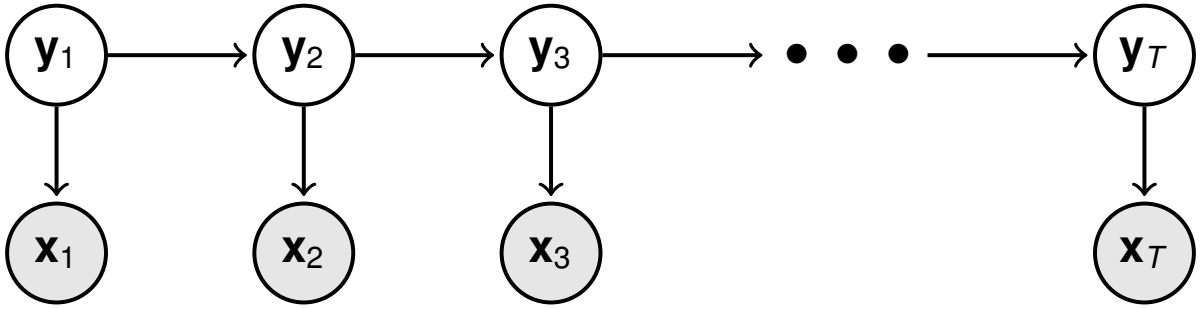
Graphical Models

Maneesh Sahani
maneesh@gatsby.ucl.ac.uk

Gatsby Computational Neuroscience Unit, and
MSc ML/CSML, Dept Computer Science
University College London

Term 1, Autumn 2014

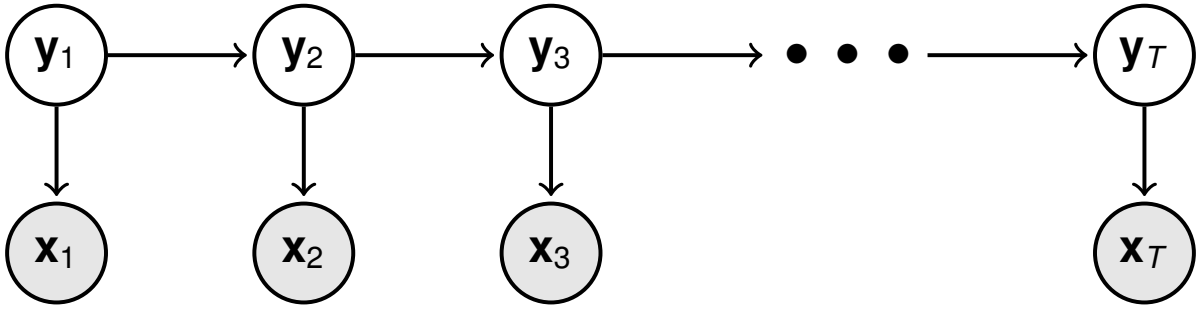
Graphs, independence and factorisation.



The (Markov) independence structure of a latent chain model implied that the joint-data likelihood factorised:

$$P(\mathcal{X}, \mathcal{Y}) = P(\mathbf{y}_1)P(\mathbf{x}_1|\mathbf{y}_1)\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t-1})P(\mathbf{x}_t|\mathbf{y}_t)$$

Graphs, independence and factorisation.

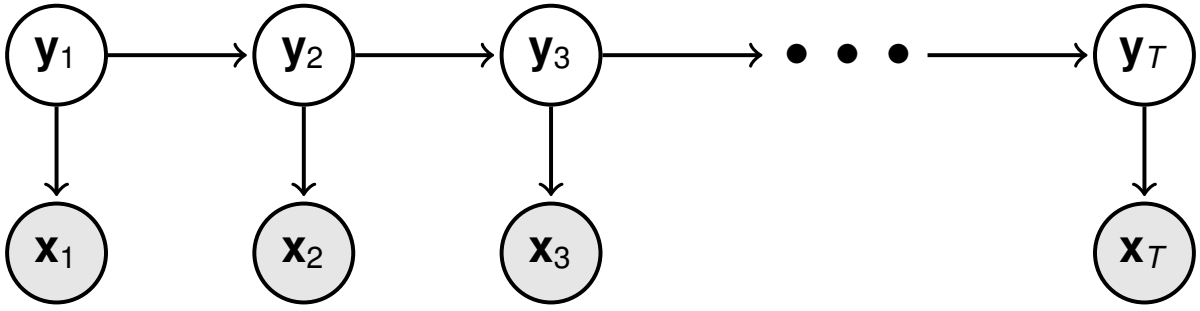


The (Markov) independence structure of a latent chain model implied that the joint-data likelihood factorised:

$$P(\mathcal{X}, \mathcal{Y}) = P(\mathbf{y}_1)P(\mathbf{x}_1|\mathbf{y}_1)\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t-1})P(\mathbf{x}_t|\mathbf{y}_t)$$

We exploited the factored form to obtain local $O(T)$ learning algorithms.

Graphs, independence and factorisation.



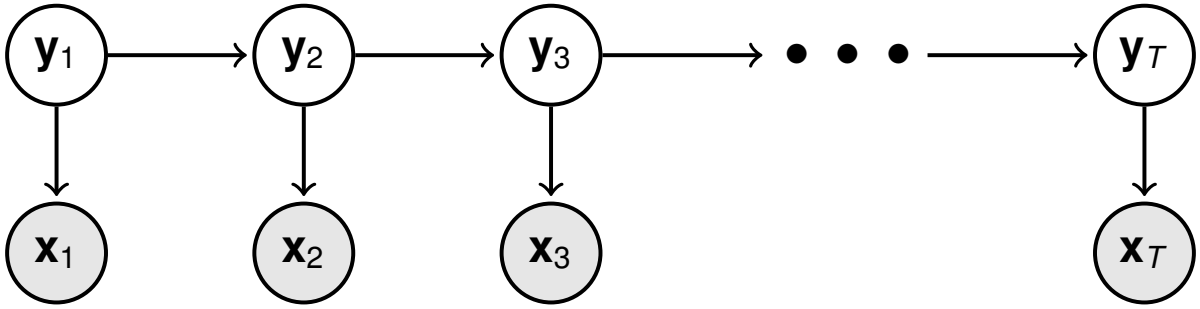
The (Markov) independence structure of a latent chain model implied that the joint-data likelihood factorised:

$$P(\mathcal{X}, \mathcal{Y}) = P(\mathbf{y}_1)P(\mathbf{x}_1|\mathbf{y}_1)\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t-1})P(\mathbf{x}_t|\mathbf{y}_t)$$

We exploited the factored form to obtain local $O(T)$ learning algorithms.

- Learning: requires only local marginals of posterior

Graphs, independence and factorisation.



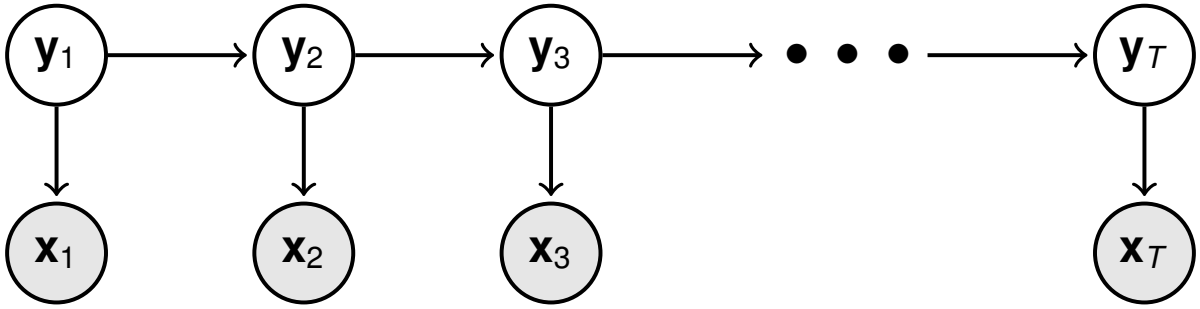
The (Markov) independence structure of a latent chain model implied that the joint-data likelihood factorised:

$$P(\mathcal{X}, \mathcal{Y}) = P(\mathbf{y}_1)P(\mathbf{x}_1|\mathbf{y}_1)\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t-1})P(\mathbf{x}_t|\mathbf{y}_t)$$

We exploited the factored form to obtain local $O(T)$ learning algorithms.

- Learning: requires only local marginals of posterior
- Inference: local marginals found by passing local messages

Graphs, independence and factorisation.



The (Markov) independence structure of a latent chain model implied that the joint-data likelihood factorised:

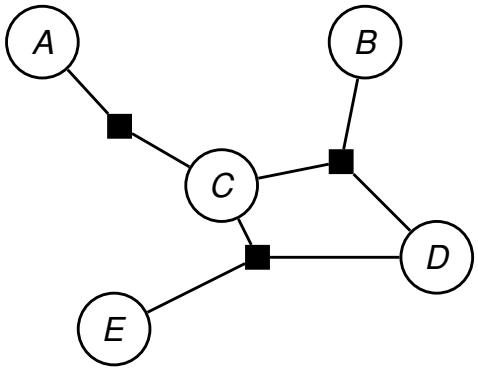
$$P(\mathcal{X}, \mathcal{Y}) = P(\mathbf{y}_1)P(\mathbf{x}_1|\mathbf{y}_1)\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t-1})P(\mathbf{x}_t|\mathbf{y}_t)$$

We exploited the factored form to obtain local $O(T)$ learning algorithms.

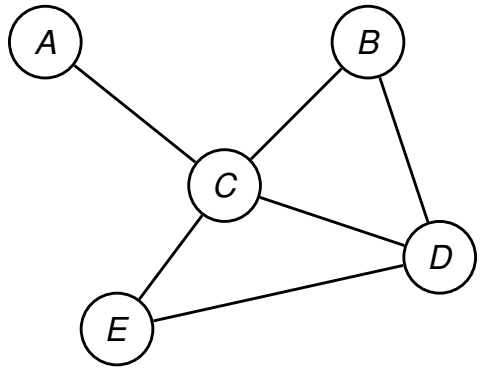
- Learning: requires only local marginals of posterior
- Inference: local marginals found by passing local messages

The independence structure of the model (and the factorisation of its likelihood) is encoded in its graph.

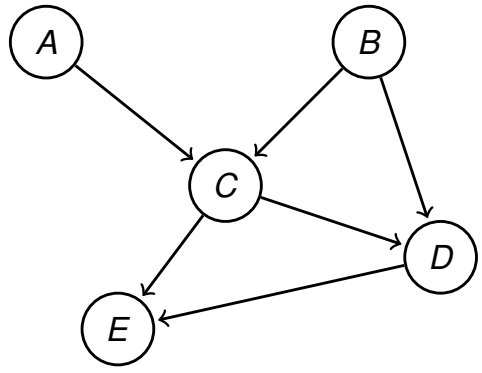
Varieties of graphical model



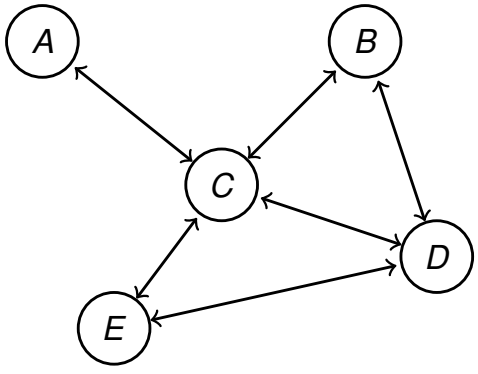
factor graph



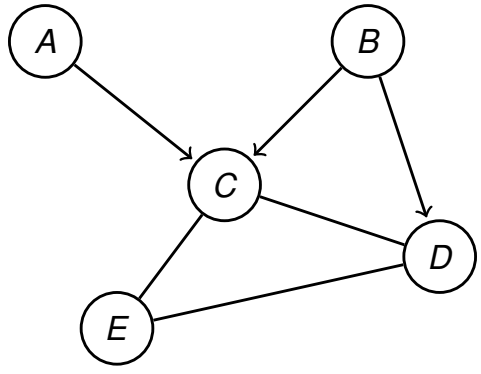
undirected graph



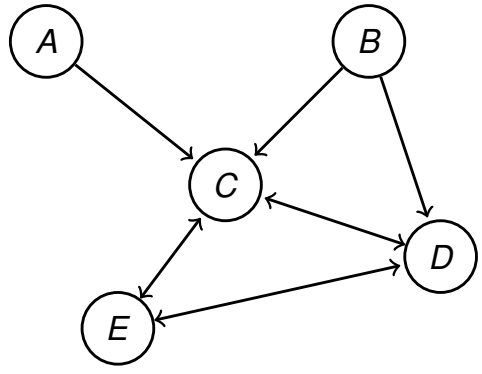
directed graph



bidirected graph



chain graph



mixed graph

- Nodes in the graph correspond to random variables.
- Edges in graph indicate statistical dependence between the variables.
- (Absent edges signal (conditional) independence between variables).

Why the graph?

- Gives an intuitive representation of the relationships amongst many variables, possibly embodying prior beliefs or knowledge about causal relationships.
  (Examples: inheritance in family trees, noise in electric circuits, neural networks)

Why the graph?

- Gives an intuitive representation of the relationships amongst many variables, possibly embodying prior beliefs or knowledge about causal relationships.
  (Examples: inheritance in family trees, noise in electric circuits, neural networks)
- Provides a precise syntax to describe these relationships, and to infer any implied (in)dependencies amongst larger groups of variables.

  Is $A \perp\!\!\!\perp E | \{B, C\}$?

Why the graph?

- Gives an intuitive representation of the relationships amongst many variables, possibly embodying prior beliefs or knowledge about causal relationships.
  (Examples: inheritance in family trees, noise in electric circuits, neural networks)
- Provides a precise syntax to describe these relationships, and to infer any implied (in)dependencies amongst larger groups of variables.

  Is $A \perp\!\!\!\perp E | \{B, C\}$?
- Each graphical structure corresponds to a parametric family of distributions that satisfy all the implied (in)dependencies.

Why the graph?

- Gives an intuitive representation of the relationships amongst many variables, possibly embodying prior beliefs or knowledge about causal relationships.
  (Examples: inheritance in family trees, noise in electric circuits, neural networks)
- Provides a precise syntax to describe these relationships, and to infer any implied (in)dependencies amongst larger groups of variables.

  Is $A \perp\!\!\!\perp E | \{B, C\}$?
- Each graphical structure corresponds to a parametric family of distributions that satisfy all the implied (in)dependencies.
- Graph-based manipulations allow us to identify the sufficient statistics of these distributions needed for learning, and to construct general-purpose message-passing algorithms that implement inference efficiently.

  Find $P(A|C = c)$ without enumerating all settings of $B, D, E \ldots$

Types of independence

For events or random variables X, Y, V:

Conditional Independence:

$$X \perp\!\!\!\perp Y | V \quad \Leftrightarrow \quad P(X|Y,V) = P(X|V) \qquad \text{[provided, for events, } P(Y,V) > 0\text{]}$$

Thus,

$$X \perp\!\!\!\perp Y | V \quad \Leftrightarrow \quad P(X,Y|V) = P(X|Y,V)P(Y|V) = P(X|V)P(Y|V)$$

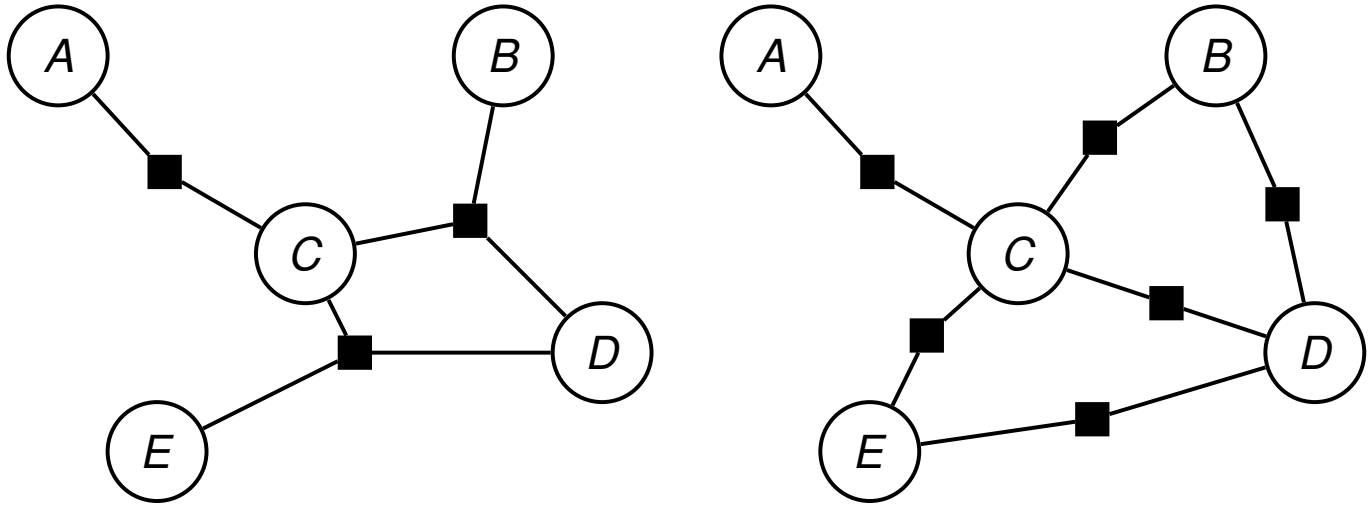
We can generalise to conditional independence between **sets of random variables**:

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{V} \quad \Leftrightarrow \quad \{X \perp\!\!\!\perp Y | \mathcal{V}, \ \ \forall X \in \mathcal{X} \ \text{ and } \ \forall Y \in \mathcal{Y}\}$$

Marginal Independence:

$$X \perp\!\!\!\perp Y \quad \Leftrightarrow \quad X \perp\!\!\!\perp Y | \varnothing \quad \Leftrightarrow \quad P(X,Y) = P(X)P(Y)$$

Factor graphs



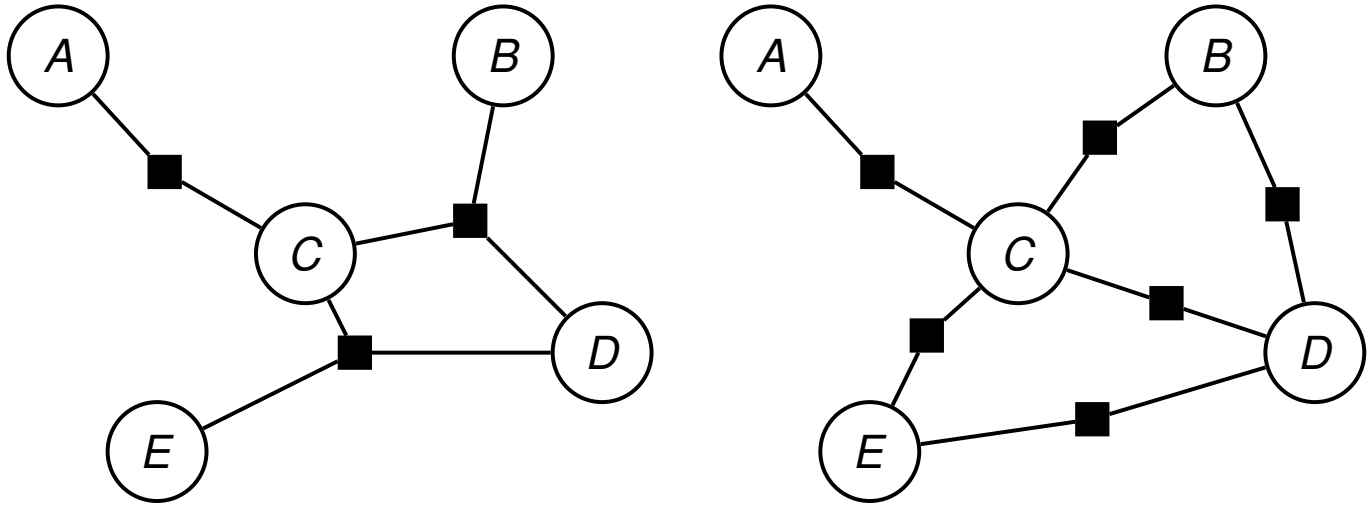
(a) (b)

A factor graph is a direct graphical representation of the factorised model structure: each square indicates a factor that depends on the linked variables.

$$P(\mathcal{X}) = \frac{1}{Z} \prod_j f_j(\mathcal{X}_{C_j})$$

where $\mathcal{X} = \{X_1, \ldots, X_K\}$, $\mathcal{X}_S = \{X_i : i \in S\}$, j indexes the factors, C_j contains the indices of variables adjacent to factor j, f_j is the factor function (also called the factor potential or clique potential) and Z is a normalisation constant.

Factor graphs: examples



(a) (b)

Examples:

(a) $P(A, B, C, D, E) = \frac{1}{Z_a} f_1(A, C) f_2(B, C, D) f_3(C, D, E)$

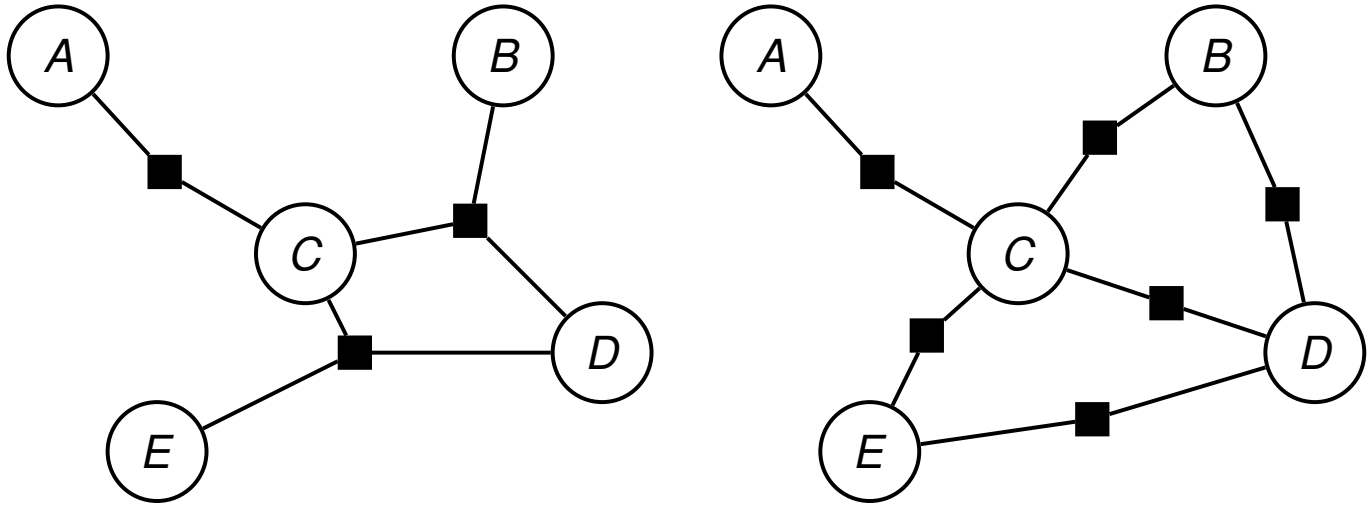
(b) $P(A, B, C, D, E) = \frac{1}{Z_b} f_1(A, C) f_2(B, C) f_3(C, D) f_4(B, D) f_5(C, E) f_6(D, E)$

and [e.g.]:

$$Z_a = \sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} \sum_{c \in \mathcal{C}} \sum_{d \in \mathcal{D}} \sum_{e \in \mathcal{E}} f_1(a, c) f_2(b, c, d) f_3(c, d, e)$$

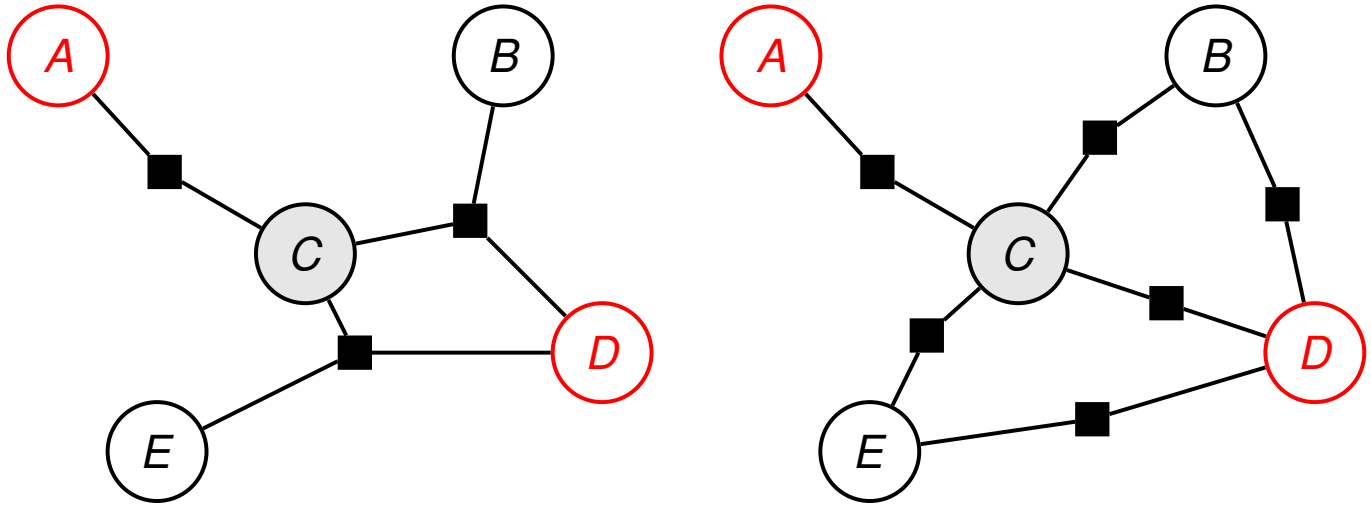
where $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$ are the domains of the corresponding random variables.

Factor graphs: conditional independence



Conditional independence: $X \perp\!\!\!\perp Y | \mathcal{V}$ if **every** path between X and Y contains some $V \in \mathcal{V}$.

Factor graphs: conditional independence

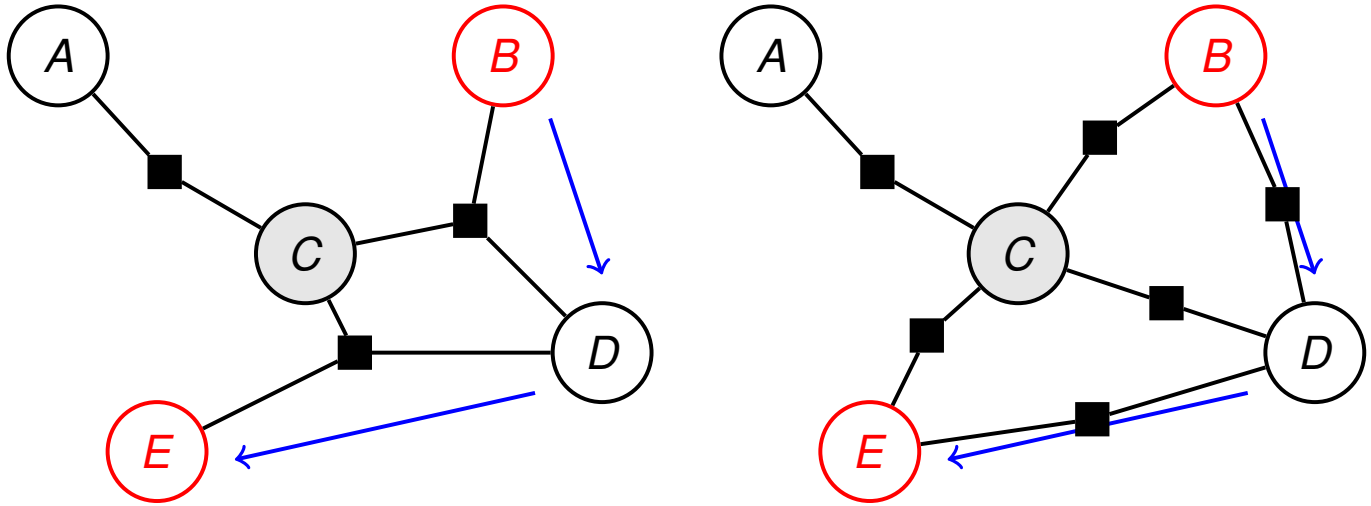


Conditional independence: $X \perp\!\!\!\perp Y | \mathcal{V}$ if **every** path between X and Y contains some $V \in \mathcal{V}$.

In both graphs:

$$A \perp\!\!\!\perp D | C$$

Factor graphs: conditional independence



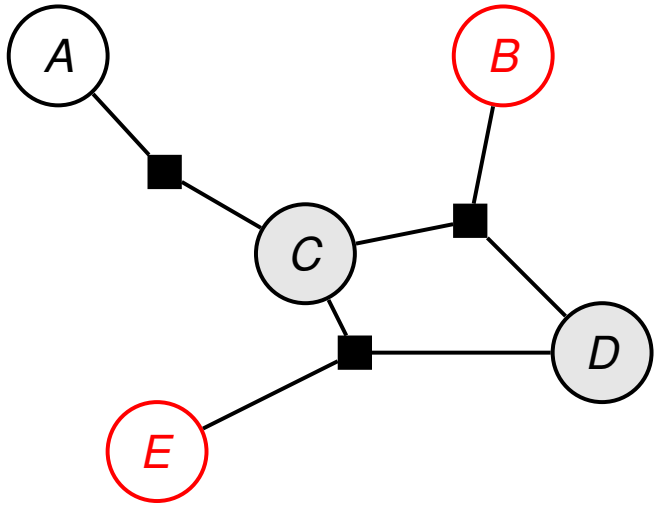
Conditional independence: $X \perp\!\!\!\perp Y | \mathcal{V}$ if **every** path between X and Y contains some $V \in \mathcal{V}$.

In both graphs:

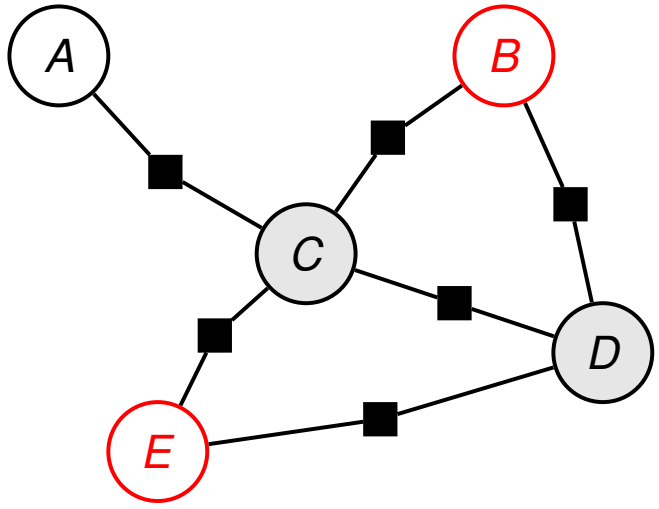
$A \perp\!\!\!\perp D | C$

$B \not\perp\!\!\!\perp E | C$

Factor graphs: conditional independence



(a)



(b)

Conditional independence: $X \perp\!\!\!\perp Y | \mathcal{V}$ if **every** path between X and Y contains some $V \in \mathcal{V}$.

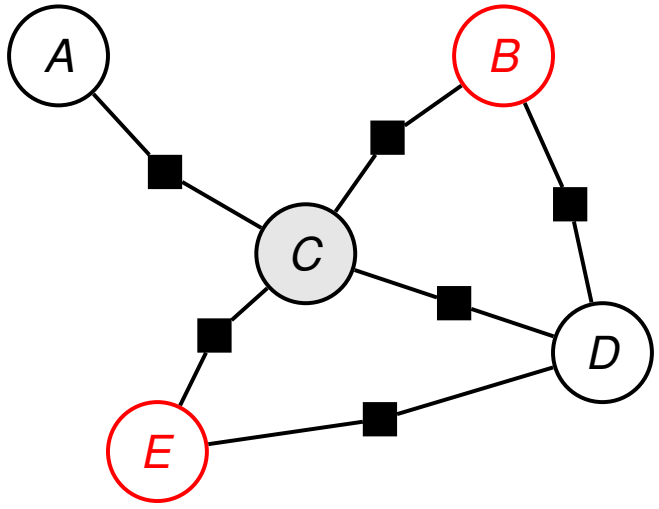
In both graphs:

$A \perp\!\!\!\perp D | C$

$B \not\perp\!\!\!\perp E | C$

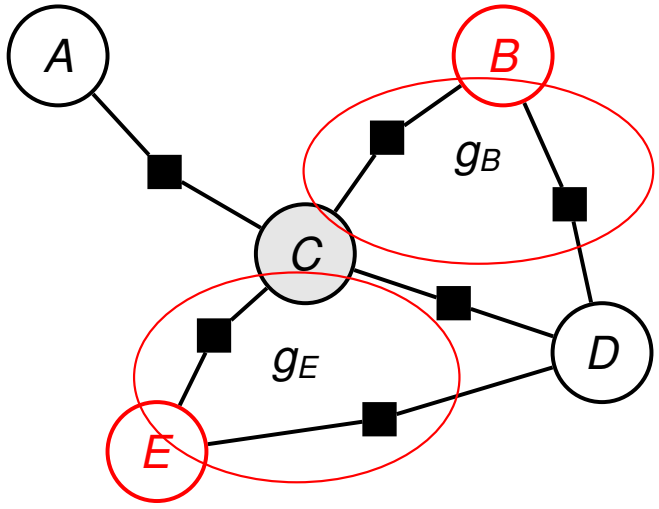
$B \perp\!\!\!\perp E | \{C, D\}$

Factorisation and conditional independence



Every path between X and Y contains some $V \in \mathcal{V}$

Factorisation and conditional independence

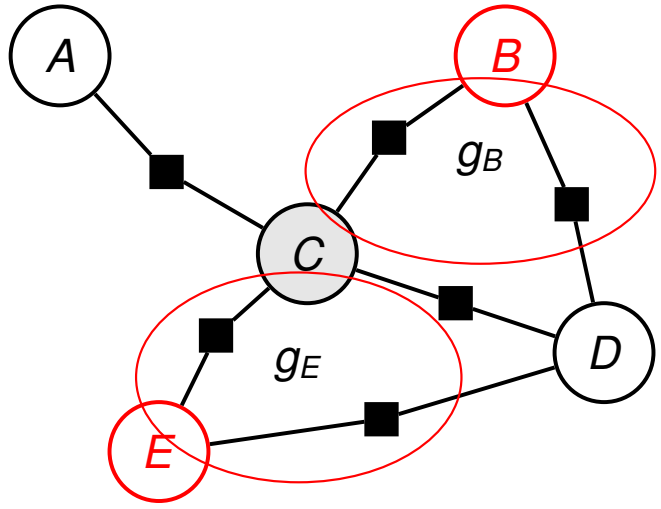


Every path between X and Y contains some $V \in \mathcal{V}$

$$\Rightarrow P(X, Y, \mathcal{V}, \dots) = \frac{1}{Z} g_X(X, \mathcal{V}_X, \mathcal{Q}_X) g_Y(Y, \mathcal{V}_Y, \mathcal{Q}_Y) g_R(\mathcal{Q}_R, \mathcal{V}_R)$$

where $\mathcal{V}_x, \mathcal{V}_y, \mathcal{V}_R \subseteq \mathcal{V}$ and the sets of remaining variables $\mathcal{Q}_X$, $\mathcal{Q}_Y$ and $\mathcal{Q}_R$ are disjoint.

Factorisation and conditional independence



Every path between X and Y contains some $V \in \mathcal{V}$

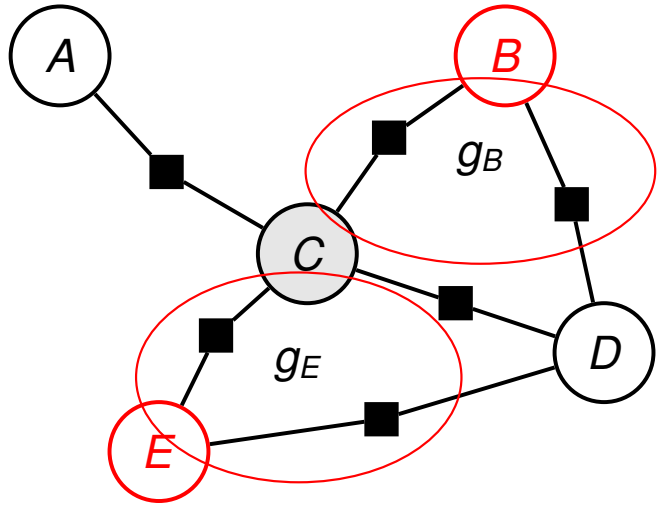
$$\Rightarrow P(X, Y, \mathcal{V}, \dots) = \frac{1}{Z} g_X(X, \mathcal{V}_X, \mathcal{Q}_X) g_Y(Y, \mathcal{V}_Y, \mathcal{Q}_Y) g_R(\mathcal{Q}_R, \mathcal{V}_R)$$

where $\mathcal{V}_x, \mathcal{V}_y, \mathcal{V}_R \subseteq \mathcal{V}$ and the sets of remaining variables $\mathcal{Q}_X$, $\mathcal{Q}_Y$ and $\mathcal{Q}_R$ are disjoint.

$$\Rightarrow P(X|Y, \mathcal{V}, \dots) = \frac{P(X, Y, \mathcal{V}, \dots)}{P(Y, \mathcal{V}, \dots)} = \frac{\frac{1}{Z} g_X(X, \mathcal{V}_X, \mathcal{Q}_X) g_Y(Y, \mathcal{V}_Y, \mathcal{Q}_Y) g_R(\mathcal{Q}_R, \mathcal{V}_R)}{\sum_{X'} \frac{1}{Z} g_X(X', \mathcal{V}_X, \mathcal{Q}_X) g_Y(Y, \mathcal{V}_Y, \mathcal{Q}_Y) g_R(\mathcal{Q}_R, \mathcal{V}_R)}$$

$$= \frac{g_X(X, \mathcal{V}_X, \mathcal{Q}_X)}{\sum_{X'} g_X(X', \mathcal{V}_X, \mathcal{Q}_X)}$$

Factorisation and conditional independence



Every path between X and Y contains some $V \in \mathcal{V}$

$$\Rightarrow P(X, Y, \mathcal{V}, \dots) = \frac{1}{Z} g_X(X, \mathcal{V}_X, \mathcal{Q}_X) g_Y(Y, \mathcal{V}_Y, \mathcal{Q}_Y) g_R(\mathcal{Q}_R, \mathcal{V}_R)$$

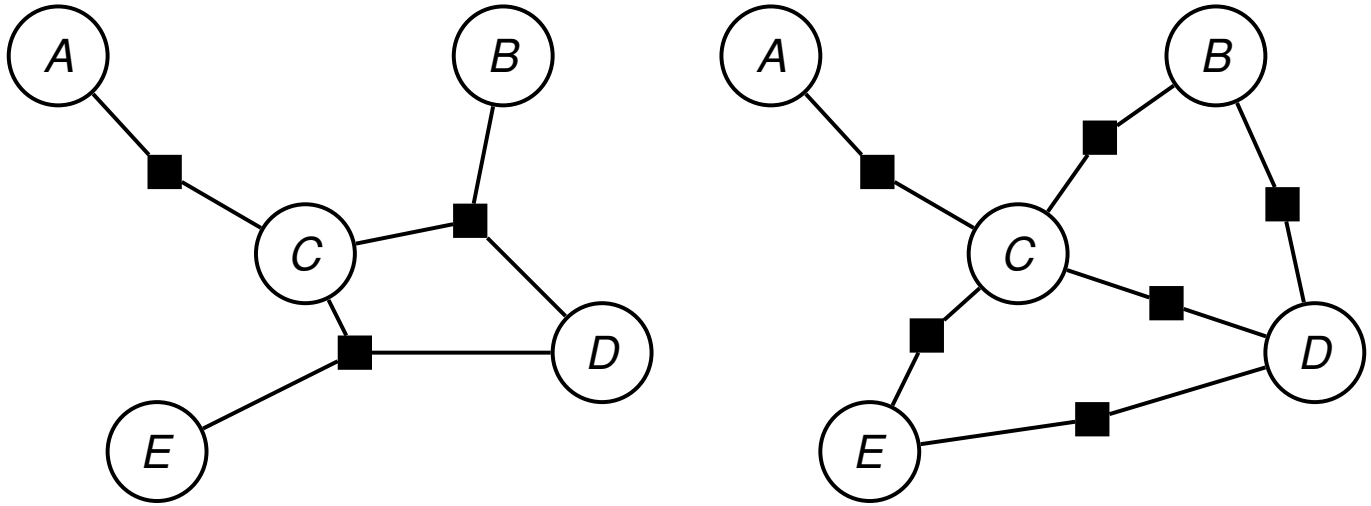
where $\mathcal{V}_x, \mathcal{V}_y, \mathcal{V}_R \subseteq \mathcal{V}$ and the sets of remaining variables $\mathcal{Q}_X$, $\mathcal{Q}_Y$ and $\mathcal{Q}_R$ are disjoint.

$$\Rightarrow P(X|Y, \mathcal{V}, \dots) = \frac{P(X, Y, \mathcal{V}, \dots)}{P(Y, \mathcal{V}, \dots)} = \frac{\frac{1}{Z} g_X(X, \mathcal{V}_X, \mathcal{Q}_X) g_Y(Y, \mathcal{V}_Y, \mathcal{Q}_Y) g_R(\mathcal{Q}_R, \mathcal{V}_R)}{\sum_{X'} \frac{1}{Z} g_X(X', \mathcal{V}_X, \mathcal{Q}_X) g_Y(Y, \mathcal{V}_Y, \mathcal{Q}_Y) g_R(\mathcal{Q}_R, \mathcal{V}_R)}$$

$$= \frac{g_X(X, \mathcal{V}_X, \mathcal{Q}_X)}{\sum_{X'} g_X(X', \mathcal{V}_X, \mathcal{Q}_X)}$$

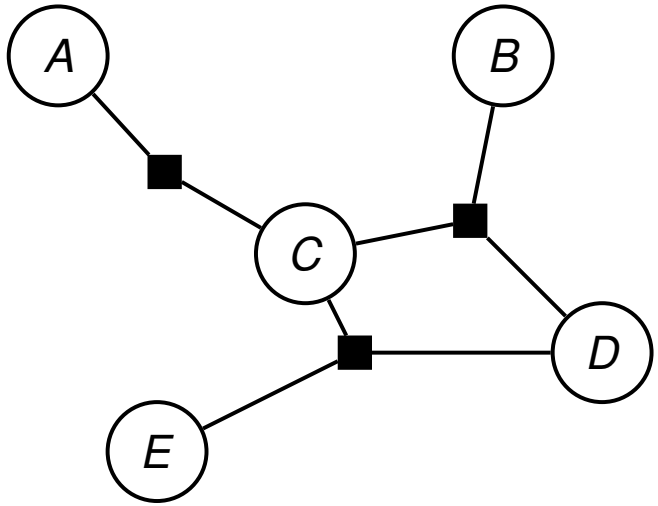
Since the RHS does not depend on Y, it follows that $X \perp\!\!\!\perp Y | \mathcal{V}$.

Factor graphs: neighbourhoods and Markov boundaries

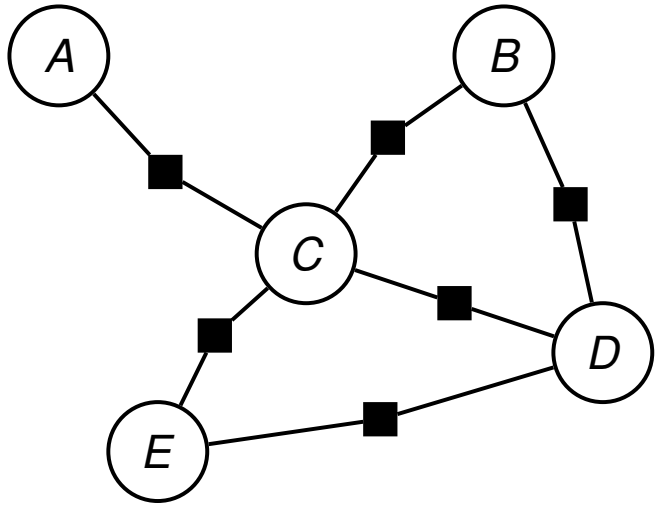


- Variables are neighbours if they share a common factor; the neighbourhood $\mathrm{ne}(X)$ is the set of all neighbours of X.

Factor graphs: neighbourhoods and Markov boundaries



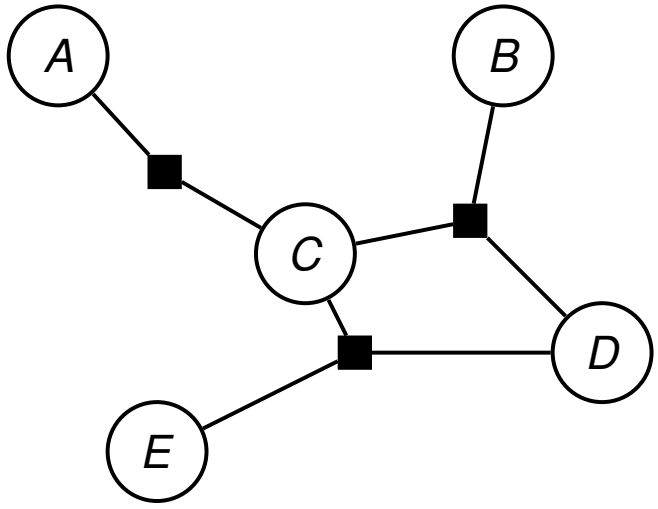
(a)



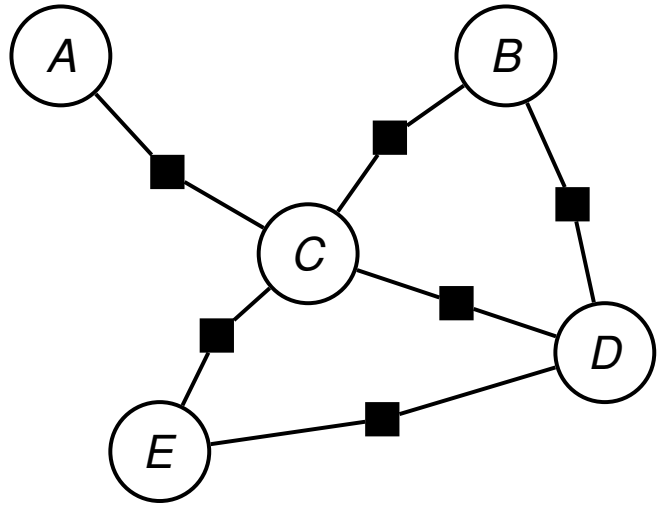
(b)

- Variables are neighbours if they share a common factor; the neighbourhood $\mathrm{ne}(X)$ is the set of all neighbours of X.
- Each variable X is **conditionally independent** of all non-neighbours given its neighbours: $X \perp\!\!\!\perp Y \,|\, \mathrm{ne}(X), \quad \forall Y \notin \{X \cup \mathrm{ne}(X)\}$
  $\Rightarrow \mathrm{ne}(X)$ is a Markov blanket for X.

Factor graphs: neighbourhoods and Markov boundaries



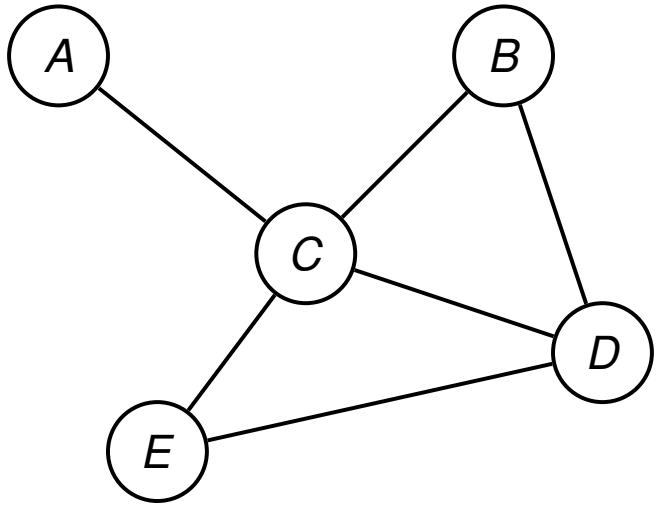
(a)



(b)

- Variables are neighbours if they share a common factor; the neighbourhood $\mathrm{ne}(X)$ is the set of all neighbours of X.
- Each variable X is **conditionally independent** of all non-neighbours given its neighbours: $X \perp\!\!\!\perp Y \,|\, \mathrm{ne}(X), \quad \forall Y \notin \{X \cup \mathrm{ne}(X)\}$
  $\Rightarrow \mathrm{ne}(X)$ is a Markov blanket for X.
- In fact, the neighbourhood is the minimal such set: the Markov boundary.

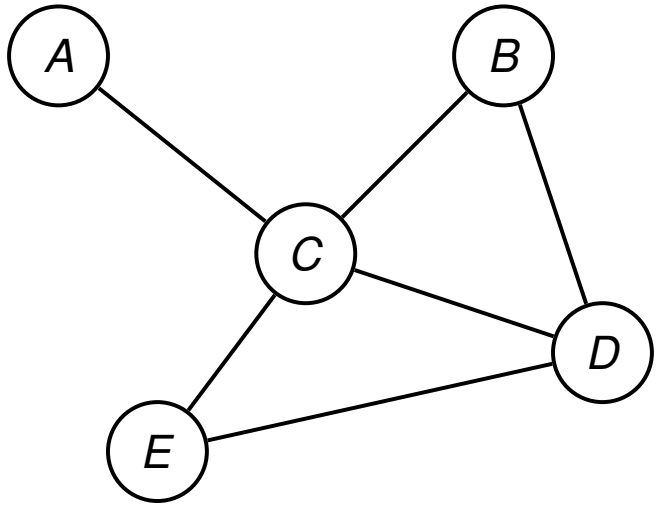
Undirected graphical models: Markov networks



An **undirected graphical model** is a direct representation of conditional independence structure. Nodes are connected iff they are conditionally dependent given all others.

$\Rightarrow$ neighbours (connected nodes) in a Markov net share a factor.

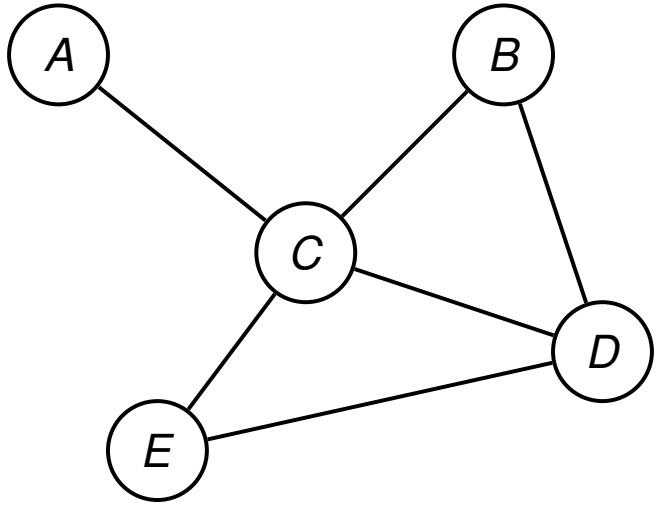
Undirected graphical models: Markov networks



An **undirected graphical model** is a direct representation of conditional independence structure. Nodes are connected iff they are conditionally dependent given all others.

- $\Rightarrow$ neighbours (connected nodes) in a Markov net share a factor.
- $\Rightarrow$ non-neighbours (disconnected nodes) in a Markov net *cannot* share a factor.

Undirected graphical models: Markov networks



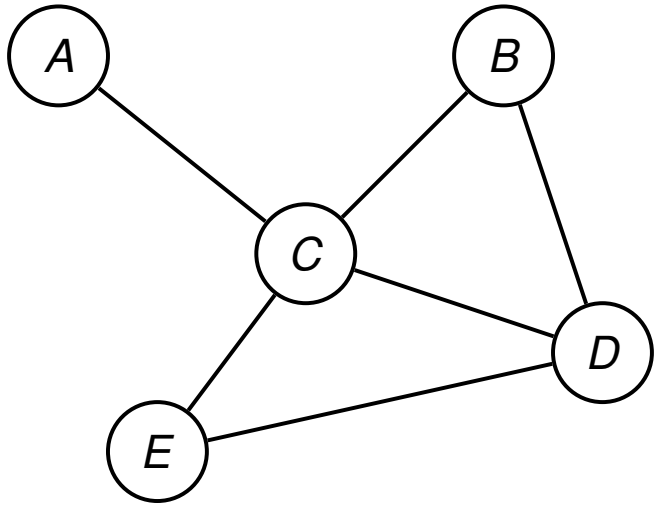
An **undirected graphical model** is a direct representation of conditional independence structure. Nodes are connected iff they are conditionally dependent given all others.

- $\Rightarrow$ neighbours (connected nodes) in a Markov net share a factor.
- $\Rightarrow$ non-neighbours (disconnected nodes) in a Markov net *cannot* share a factor.
- $\Rightarrow$ the joint probability factors over the **maximal cliques** C_j of the graph:

$$P(\mathcal{X}) = \frac{1}{Z} \prod_j f_j(\mathcal{X}_{C_j})$$

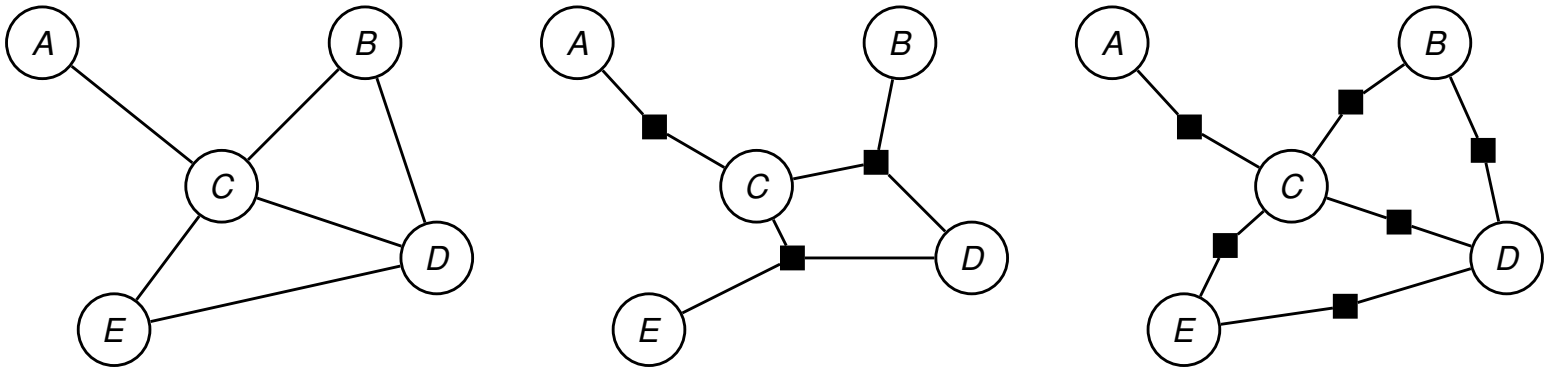
[Cliques are fully connected subgraphs, maximal cliques are cliques not contained in other cliques.]

Undirected graphs: Markov boundaries



- $X \perp\!\!\!\perp Y | \mathcal{V}$ if every path between X and Y contains some node $V \in \mathcal{V}$
- Each variable X is conditionally independent of all non-neighbours given its neighbours: $X \perp\!\!\!\perp Y | \text{ne}(X), \quad \forall Y \notin \{X \cup \text{ne}(X)\}$
- $\mathcal{V}$ is a **Markov blanket** for X iff $X \perp\!\!\!\perp Y | \mathcal{V}$ for all $Y \notin \{X \cup \mathcal{V}\}$.
- **Markov boundary:** minimal Markov blanket. For undirected graphs (like factor graphs) this is the set of neighbours of X.

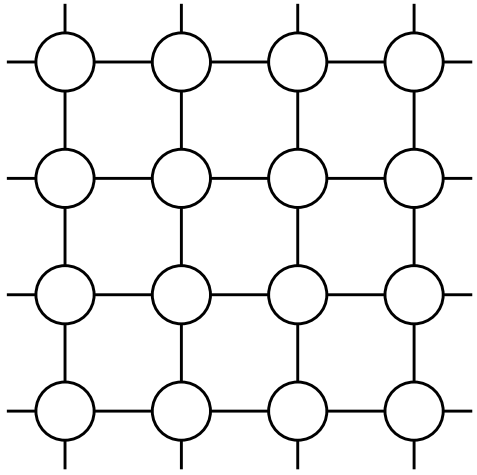
Undirected graphs and factor graphs



- Each node has the same neighbours in each graph, so (a), (b) and (c) represent exactly the same conditional independence relationships.
- The implied maximal factorisations differ: (b) has two three-way factors; (c) has only pairwise factors; (a) cannot distinguish between these (so we have to adopt factorisation (b) to be safe).
- Suppose all variables are discrete and can take on K possible values. Then the functions in (a) and (b) are tables with $\mathcal{O}(K^3)$ cells, whereas in (c) they are $\mathcal{O}(K^2)$.
- Factor graphs have richer expressive power than undirected graphical models.
- Factors cannot be determined solely by testing for conditional independence.

Some examples of undirected graphical models

- Markov random fields (used in computer vision)

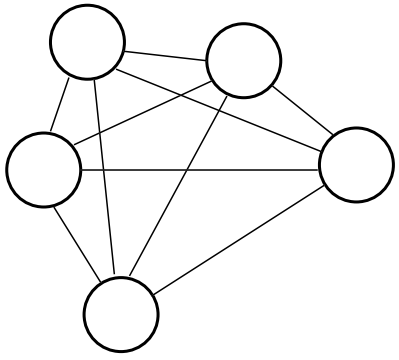


- Maximum entropy language models (used in speech and language modelling)

$$P(\mathcal{X}) = \frac{1}{Z} p_0(\mathcal{X}) \exp\left\{\sum_j \lambda_j g_j(\mathcal{X})\right\}$$

- Conditional random fields are undirected graphical models (conditioned on the input variables).

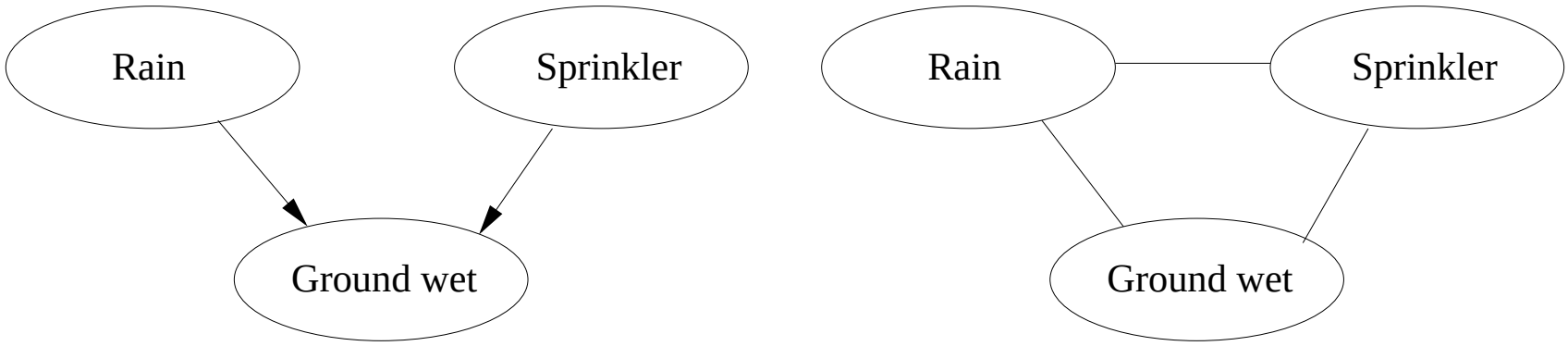
- Boltzmann machines (a kind of neural network/Ising model)



Limitations of undirected and factor graphs

Undirected and factor graphs fail to capture some useful independencies—a pair of variables may be connected merely because some other variable depends on them:

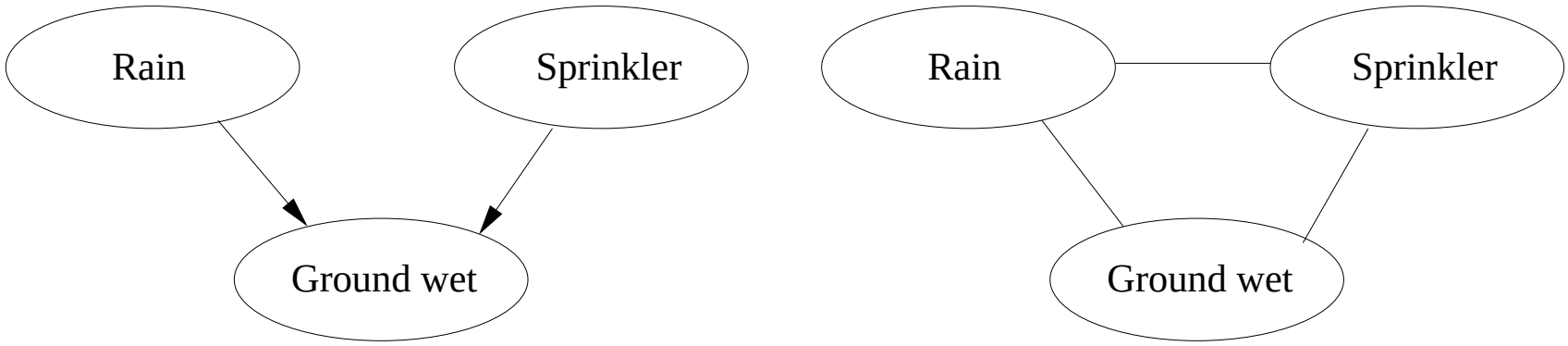
The classic example (due to Pearl):



Limitations of undirected and factor graphs

Undirected and factor graphs fail to capture some useful independencies—a pair of variables may be connected merely because some other variable depends on them:

The classic example (due to Pearl):

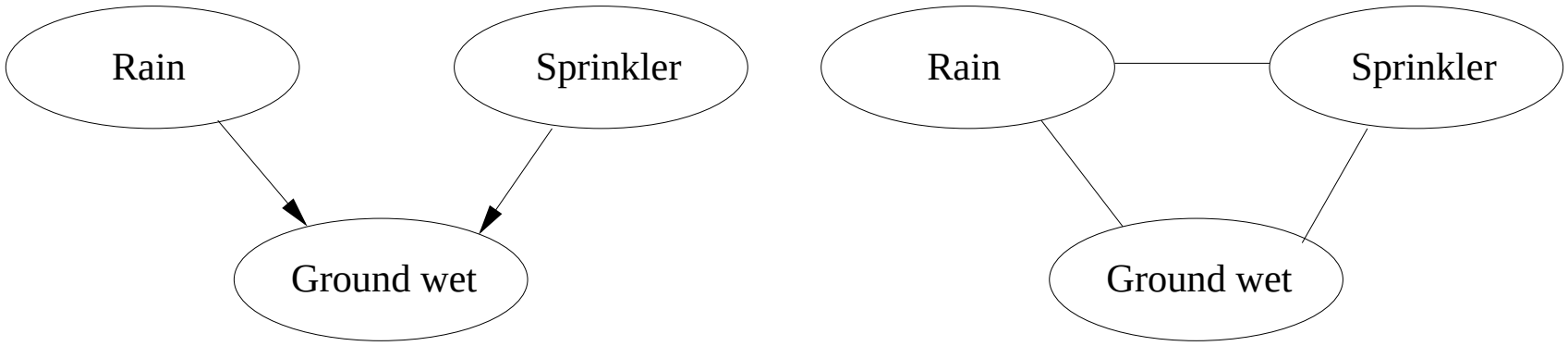


- Most sprinklers switch on come rain or shine; and certainly the weather pays no heed to the state of the spinklers.

Limitations of undirected and factor graphs

Undirected and factor graphs fail to capture some useful independencies—a pair of variables may be connected merely because some other variable depends on them:

The classic example (due to Pearl):

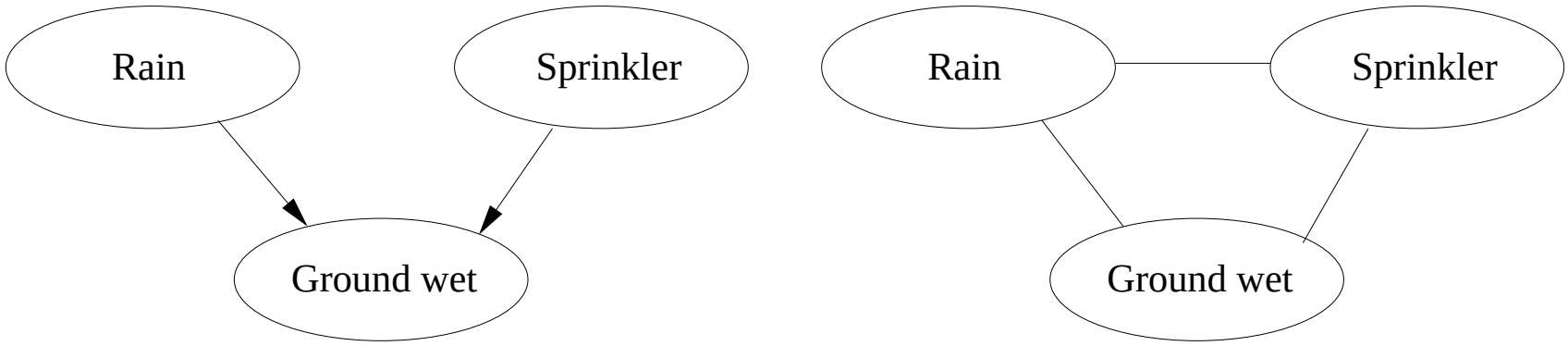


- Most sprinklers switch on come rain or shine; and certainly the weather pays no heed to the state of the spinklers.
- **Explaining away**: Damp ground suggests that it has rained; but if we also see a running sprinkler this explains away the damp, returning our belief about rain to the prior.

Limitations of undirected and factor graphs

Undirected and factor graphs fail to capture some useful independencies—a pair of variables may be connected merely because some other variable depends on them:

The classic example (due to Pearl):

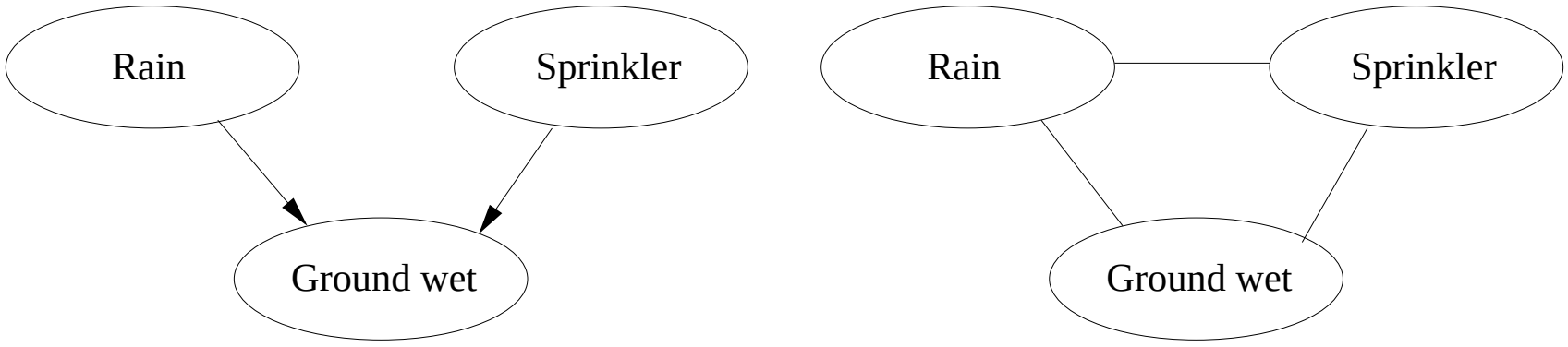


- Most sprinklers switch on come rain or shine; and certainly the weather pays no heed to the state of the spinklers.
- **Explaining away**: Damp ground suggests that it has rained; but if we also see a running sprinkler this explains away the damp, returning our belief about rain to the prior.
- $R \perp\!\!\!\perp S | \varnothing$ but $R \not\perp\!\!\!\perp S | G$.

Limitations of undirected and factor graphs

Undirected and factor graphs fail to capture some useful independencies—a pair of variables may be connected merely because some other variable depends on them:

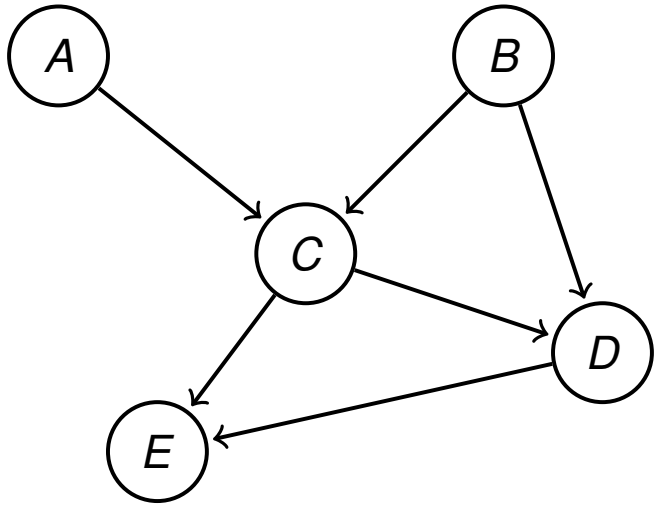
The classic example (due to Pearl):



- Most sprinklers switch on come rain or shine; and certainly the weather pays no heed to the state of the spinklers.
- **Explaining away**: Damp ground suggests that it has rained; but if we also see a running sprinkler this explains away the damp, returning our belief about rain to the prior.
- $R \perp\!\!\!\perp S | \varnothing$ but $R \not\perp\!\!\!\perp S | G$.

This highlights the difference between **marginal** and **conditional independence**.

Directed acyclic graphical models



A **directed acyclic graphical (DAG) model** represents a factorization of the joint probability distribution in terms of conditionals:

$$P(A, B, C, D, E) = P(A)P(B)P(C|A, B)P(D|B, C)P(E|C, D)$$

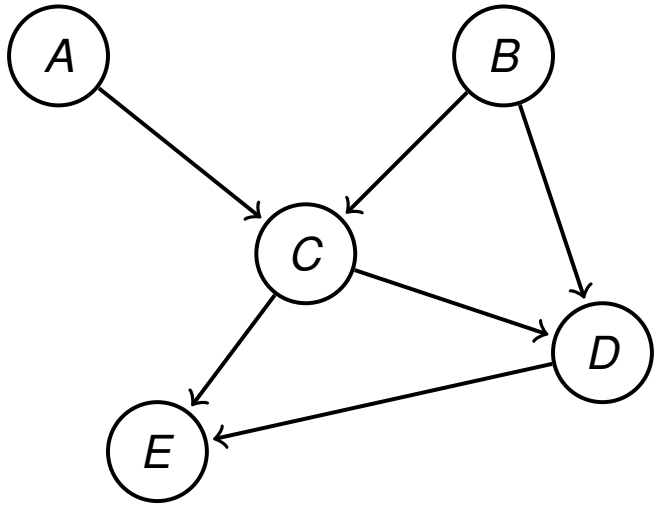
In general:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|X_{\text{pa}(i)})$$

where $\text{pa}(i)$ are the parents of node i.

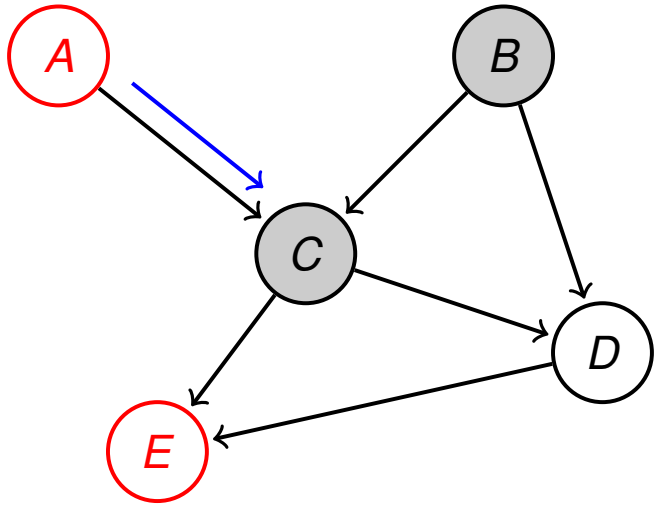
DAG models are also known as **Bayesian networks** or **Bayes net**.

Conditional independence in DAGs



Reading conditional independence from DAGs is more complicated than in undirected graphs.

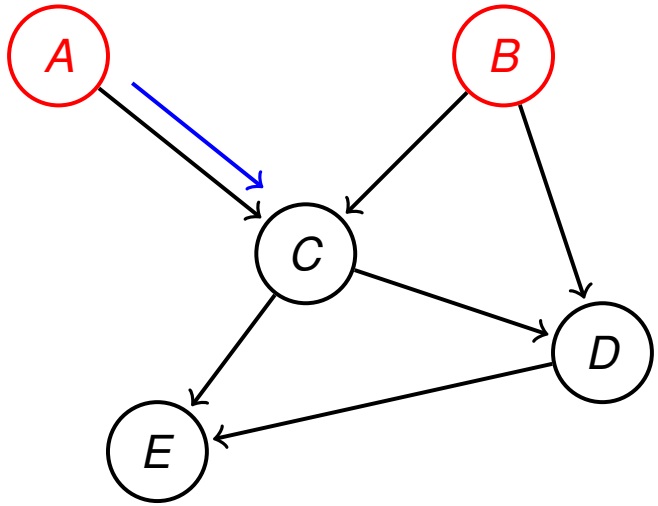
Conditional independence in DAGs



Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: conditioning nodes block paths

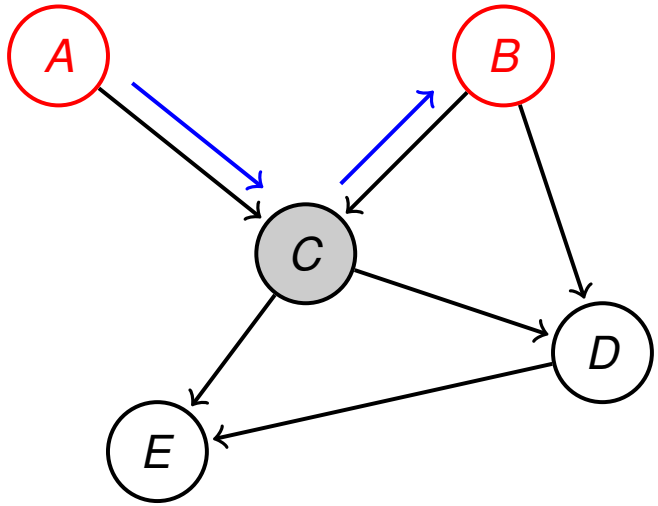
Conditional independence in DAGs



Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: conditioning nodes block paths
- $A \perp\!\!\!\perp B \mid \varnothing$: *other* nodes block reflected paths

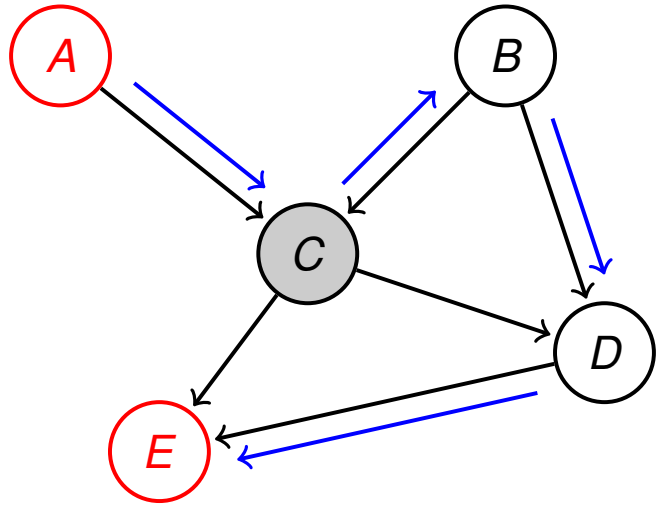
Conditional independence in DAGs



Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: conditioning nodes block paths
- $A \perp\!\!\!\perp B \mid \varnothing$: *other* nodes block reflected paths
- $A \not\perp\!\!\!\perp B \mid C$: conditioning node *creates* a reflected path by explaining away

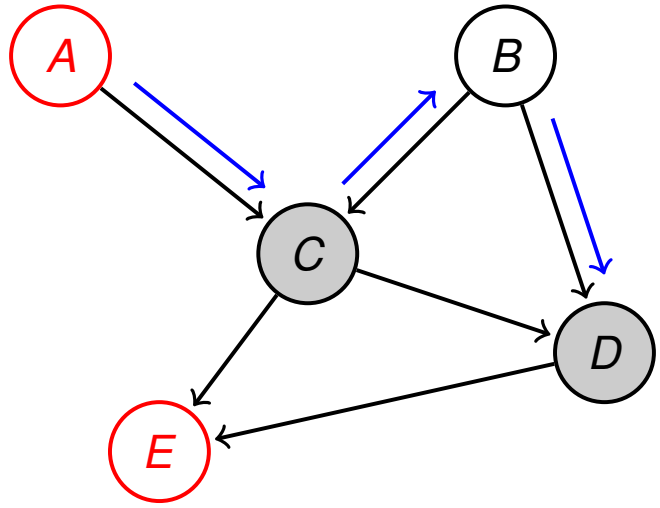
Conditional independence in DAGs



Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: conditioning nodes block paths
- $A \perp\!\!\!\perp B \mid \varnothing$: *other* nodes block reflected paths
- $A \not\perp\!\!\!\perp B \mid C$: conditioning node *creates* a reflected path by explaining away
- $A \not\perp\!\!\!\perp E \mid C$: the created path extends to E via D

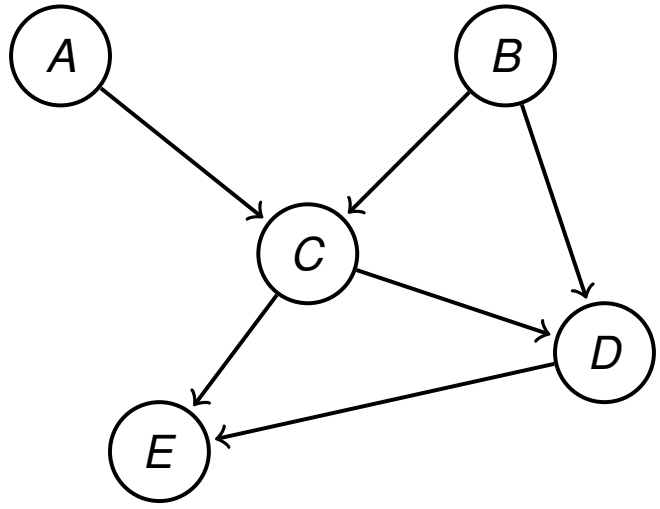
Conditional independence in DAGs



Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: conditioning nodes block paths
- $A \perp\!\!\!\perp B \mid \varnothing$: *other* nodes block reflected paths
- $A \not\perp\!\!\!\perp B \mid C$: conditioning node *creates* a reflected path by explaining away
- $A \not\perp\!\!\!\perp E \mid C$: the created path extends to E via D
- $A \perp\!\!\!\perp E \mid \{C, D\}$: but is blocked by observing D

Conditional independence in DAGs

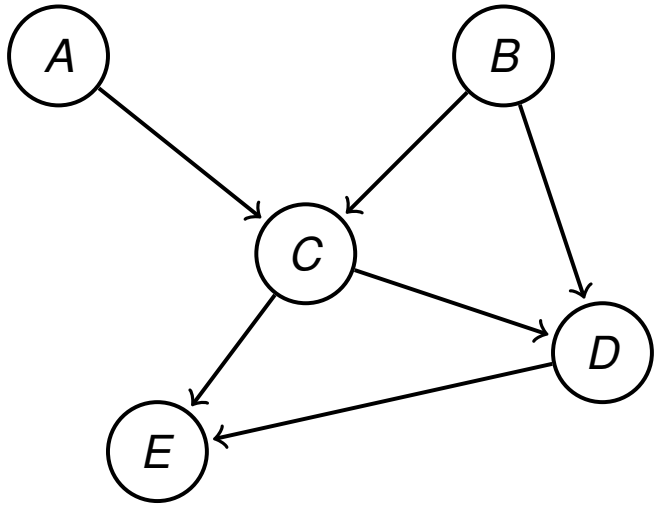


Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: conditioning nodes block paths
- $A \perp\!\!\!\perp B \mid \varnothing$: *other* nodes block reflected paths
- $A \not\perp\!\!\!\perp B \mid C$: conditioning node *creates* a reflected path by explaining away
- $A \not\perp\!\!\!\perp E \mid C$: the created path extends to E via D
- $A \perp\!\!\!\perp E \mid \{C, D\}$: but is blocked by observing D

So conditioning on (i.e. observing) nodes can both create and remove dependencies.

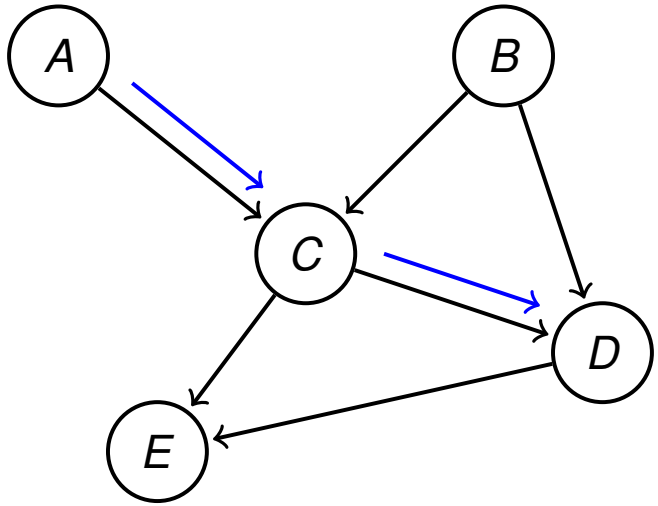
The Bayes-ball algorithm



Game: can you get a ball from X to Y without being blocked by $\mathcal{V}$? If so, $X \not\perp\!\!\!\perp Y | \mathcal{V}$.

Rules: ball follow edges, and are passed on or bounced back from nodes according to:

The Bayes-ball algorithm

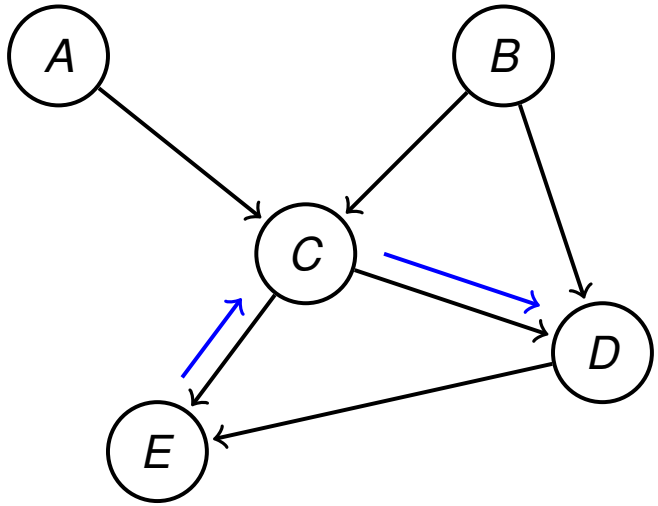


Game: can you get a ball from X to Y without being blocked by $\mathcal{V}$? If so, $X \not\perp\!\!\!\perp Y | \mathcal{V}$.

Rules: ball follow edges, and are passed on or bounced back from nodes according to:

- Nodes $V \notin \mathcal{V}$ pass balls down or up chains: $\rightarrow V \rightarrow$ or $\leftarrow V \leftarrow$.

The Bayes-ball algorithm

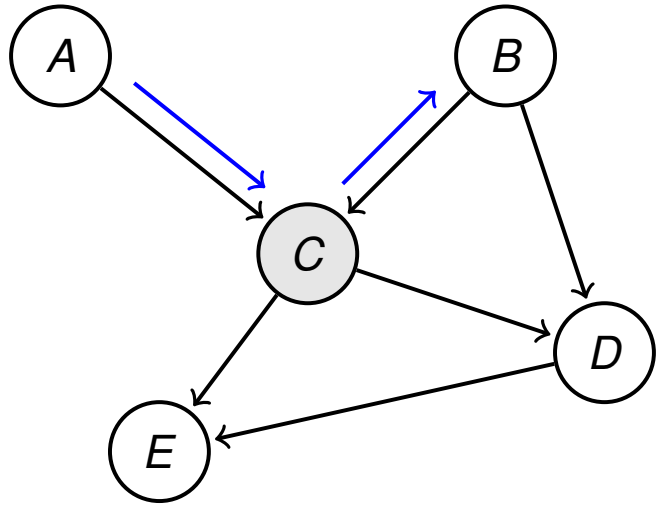


Game: can you get a ball from X to Y without being blocked by $\mathcal{V}$? If so, $X \not\perp\!\!\!\perp Y | \mathcal{V}$.

Rules: ball follow edges, and are passed on or bounced back from nodes according to:

- Nodes $V \notin \mathcal{V}$ pass balls down or up chains: $\rightarrow V \rightarrow$ or $\leftarrow V \leftarrow$.
- Nodes $V \notin \mathcal{V}$, bounce balls **from** children **to** children.

The Bayes-ball algorithm

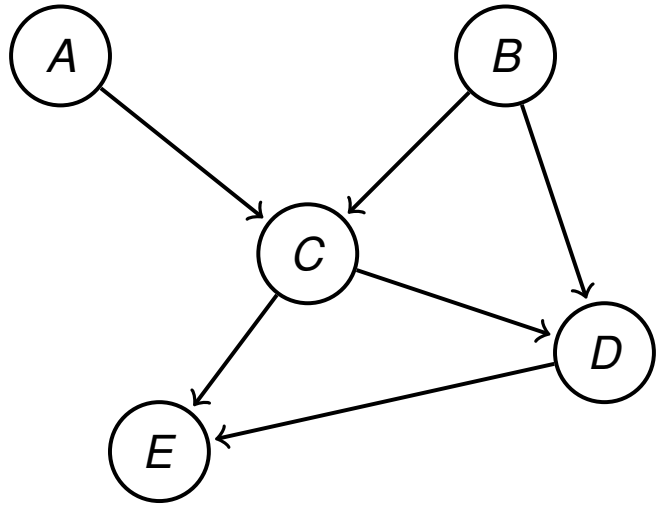


Game: can you get a ball from X to Y without being blocked by $\mathcal{V}$? If so, $X \not\perp\!\!\!\perp Y | \mathcal{V}$.

Rules: ball follow edges, and are passed on or bounced back from nodes according to:

- Nodes $V \notin \mathcal{V}$ pass balls down or up chains: $\rightarrow V \rightarrow$ or $\leftarrow V \leftarrow$.
- Nodes $V \notin \mathcal{V}$, bounce balls **from** children **to** children.
- Nodes $V \in \mathcal{V}$, bounce balls **from** parents **to** parents (including returning the ball whence it came).

The Bayes-ball algorithm



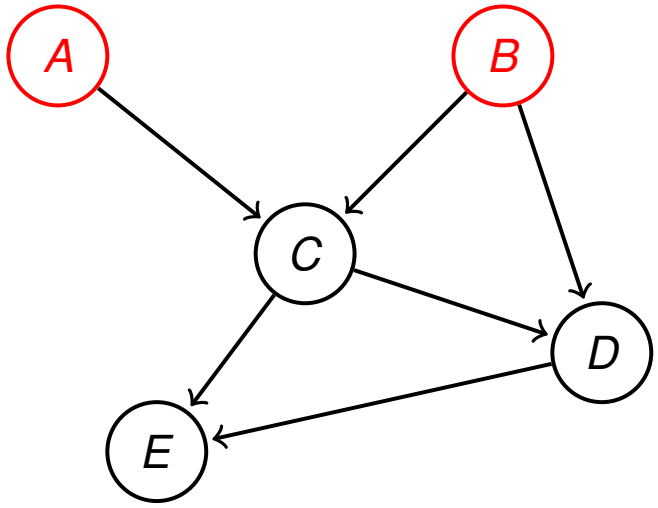
Game: can you get a ball from X to Y without being blocked by $\mathcal{V}$? If so, $X \not\perp\!\!\!\perp Y | \mathcal{V}$.

Rules: ball follow edges, and are passed on or bounced back from nodes according to:

- Nodes $V \notin \mathcal{V}$ pass balls down or up chains: $\rightarrow V \rightarrow$ or $\leftarrow V \leftarrow$.
- Nodes $V \notin \mathcal{V}$, bounce balls **from** children **to** children.
- Nodes $V \in \mathcal{V}$, bounce balls **from** parents **to** parents (including returning the ball whence it came).

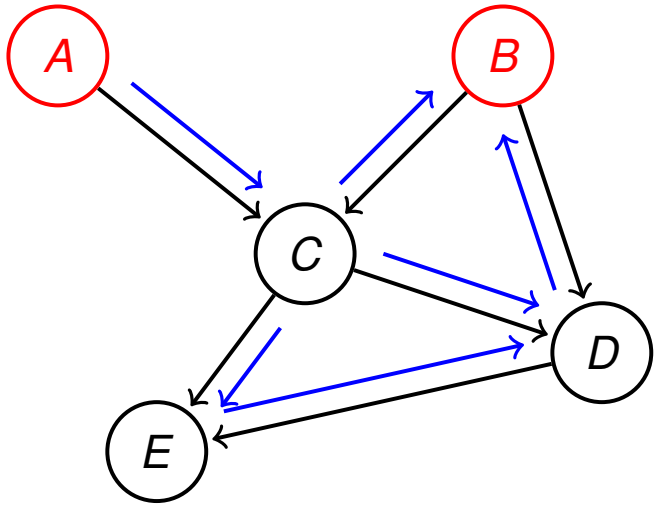
Otherwise the ball is blocked. (So $V \in \mathcal{V}$ blocks all balls from children, and stops balls from parents reaching children.)

D-separation



So when is $X \perp\!\!\!\perp Y | \mathcal{V}$?

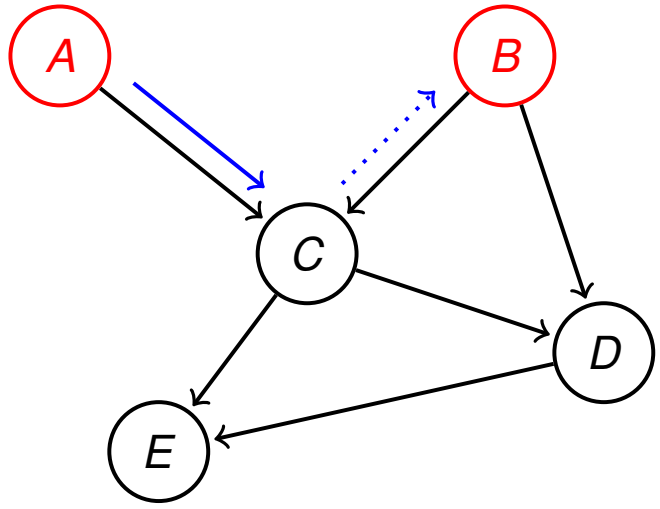
D-separation



So when is $X \perp\!\!\!\perp Y | \mathcal{V}$?

Consider *every* undirected path (i.e. ignoring arrows) between X and Y. The path is **blocked** by $\mathcal{V}$ if there is a node V on the path such that either:

D-separation

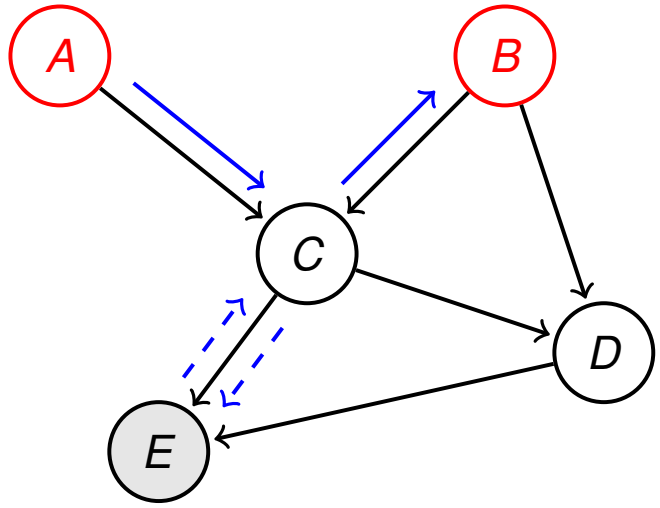


So when is $X \perp\!\!\!\perp Y | \mathcal{V}$?

Consider *every* undirected path (i.e. ignoring arrows) between X and Y. The path is **blocked** by $\mathcal{V}$ if there is a node V on the path such that either:

- V has convergent arrows ($\rightarrow V \leftarrow$) on the path (*i.e.*, V is a "collider node") and **neither V nor its descendents** $\in \mathcal{V}$.

D-separation

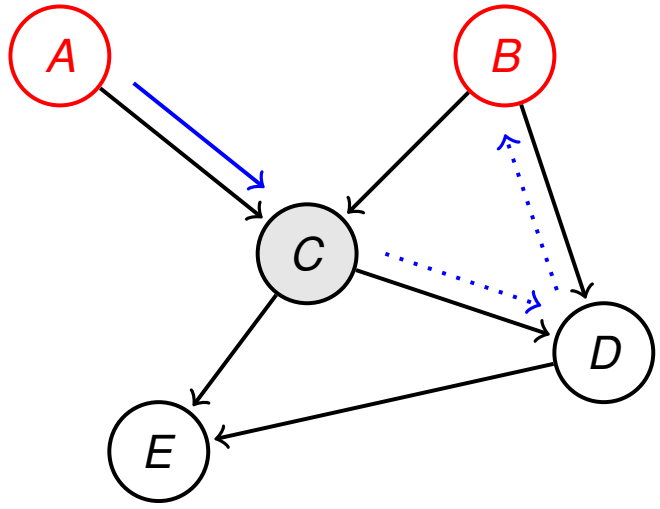


So when is $X \perp\!\!\!\perp Y | \mathcal{V}$?

Consider *every* undirected path (i.e. ignoring arrows) between X and Y. The path is **blocked** by $\mathcal{V}$ if there is a node V on the path such that either:

- V has convergent arrows ($\rightarrow V \leftarrow$) on the path (*i.e.*, V is a "collider node") and **neither V nor its descendents** $\in \mathcal{V}$.

D-separation

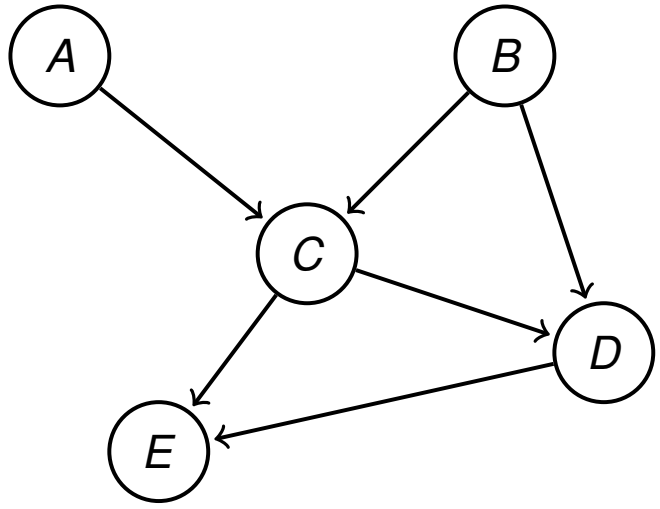


So when is $X \perp\!\!\!\perp Y | \mathcal{V}$?

Consider *every* undirected path (i.e. ignoring arrows) between X and Y. The path is **blocked** by $\mathcal{V}$ if there is a node V on the path such that either:

- V has convergent arrows ($\rightarrow V \leftarrow$) on the path (*i.e.*, V is a "collider node") and **neither V nor its descendents** $\in \mathcal{V}$.
- V does not have convergent arrows on the path ($\rightarrow V \rightarrow$ or $\leftarrow V \rightarrow$) and $V \in \mathcal{V}$. This is similar to the undirected graph semantics.

D-separation



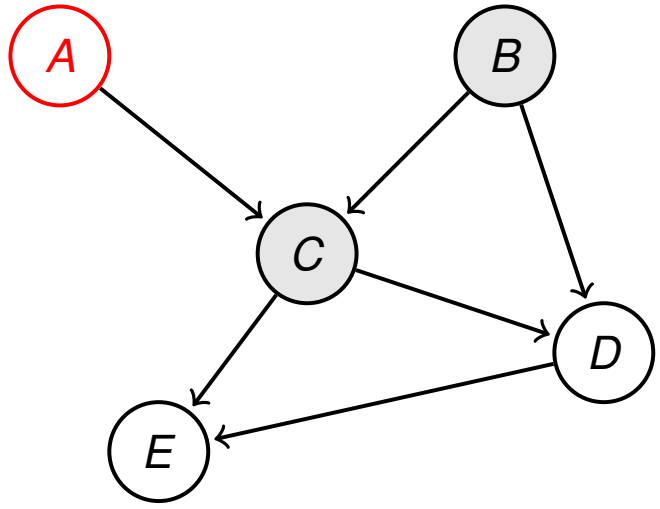
So when is $X \perp\!\!\!\perp Y | \mathcal{V}$?

Consider *every* undirected path (i.e. ignoring arrows) between X and Y. The path is **blocked** by $\mathcal{V}$ if there is a node V on the path such that either:

- V has convergent arrows ($\rightarrow V \leftarrow$) on the path (*i.e.*, V is a "collider node") and **neither V nor its descendents** $\in \mathcal{V}$.
- V does not have convergent arrows on the path ($\rightarrow V \rightarrow$ or $\leftarrow V \rightarrow$) and $V \in \mathcal{V}$. This is similar to the undirected graph semantics.

If all paths are blocked, we say $\mathcal{V}$ **d-separates** X from Y (d for directed), and $X \perp\!\!\!\perp Y | \mathcal{V}$.

D-separation



So when is $X \perp\!\!\!\perp Y | \mathcal{V}$?

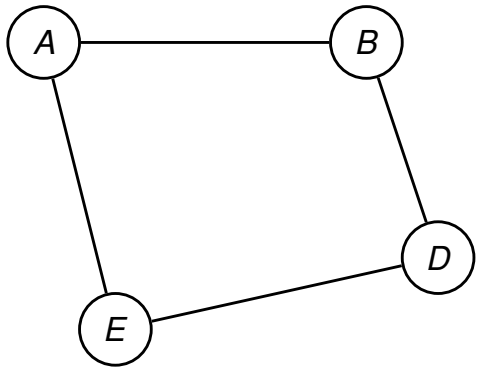
Consider *every* undirected path (i.e. ignoring arrows) between X and Y. The path is **blocked** by $\mathcal{V}$ if there is a node V on the path such that either:

- V has convergent arrows ($\rightarrow V \leftarrow$) on the path (*i.e.*, V is a "collider node") and **neither V nor its descendents** $\in \mathcal{V}$.
- V does not have convergent arrows on the path ($\rightarrow V \rightarrow$ or $\leftarrow V \rightarrow$) and $V \in \mathcal{V}$. This is similar to the undirected graph semantics.

If all paths are blocked, we say $\mathcal{V}$ **d-separates** X from Y (d for directed), and $X \perp\!\!\!\perp Y | \mathcal{V}$.

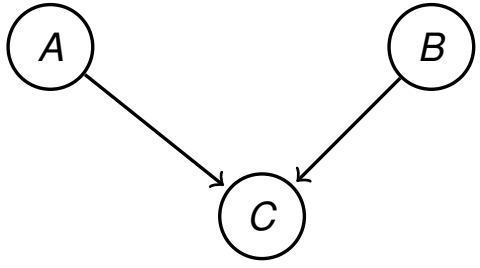
Markov boundary for X: $\{\text{pa}(X) \cup \text{ch}(X) \cup \text{pa}(\text{ch}(X))\}$.

Expressive power of directed and undirected graphs



No DAG can represent these and only these independencies

No matter how we direct the arrows there will always be two non-adjacent parents sharing a common child $\Rightarrow$ dependence in DAG but independence in undirected graph.



No undirected or factor graph can represent these and only these independencies

One three-way factor, but this does not encode marginal independence.

Graphs, conditional independencies, and families of distributions

Each graph G implies a set of conditional independence statements $\mathcal{C}(G) = \{X_i \perp\!\!\!\perp Y_i | \mathcal{V}_i\}$.

Each such set $\mathcal{C}$, defines a family of distributions that satisfy all the statements in $\mathcal{C}$:

$$\mathcal{P}_{\mathcal{C}(G)} = \{P(\mathcal{X}) \ : \ P(X_i, Y_i|\mathcal{V}_i) = P(X_i|\mathcal{V}_i)P(Y_i|\mathcal{V}_i) \text{ for all } X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i \text{ in } \mathcal{C} \ \}$$

G may also encode a family of distributions by their functional form, e.g. for a factor graph

$$\mathcal{P}_G = \{P(\mathcal{X}) \ : \ P(\mathcal{X}) = \tfrac{1}{Z} \textstyle\prod_j f_j(\mathcal{X}_{C_j}), \text{ for some non-negative functions } f_j \ \}$$

Graphs, conditional independencies, and families of distributions

Each graph G implies a set of conditional independence statements $\mathcal{C}(G) = \{X_i \perp\!\!\!\perp Y_i | \mathcal{V}_i\}$.

Each such set $\mathcal{C}$, defines a family of distributions that satisfy all the statements in $\mathcal{C}$:

$$\mathcal{P}_{\mathcal{C}(G)} = \{P(\mathcal{X}) \ : \ P(X_i, Y_i|\mathcal{V}_i) = P(X_i|\mathcal{V}_i)P(Y_i|\mathcal{V}_i) \text{ for all } X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i \text{ in } \mathcal{C} \ \}$$

G may also encode a family of distributions by their functional form, e.g. for a factor graph

$$\mathcal{P}_G = \{P(\mathcal{X}) \ : \ P(\mathcal{X}) = \tfrac{1}{Z}\prod_j f_j(\mathcal{X}_{C_j}), \text{ for some non-negative functions } f_j \ \}$$

- For directed graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$.

Graphs, conditional independencies, and families of distributions

Each graph G implies a set of conditional independence statements $\mathcal{C}(G) = \{X_i \perp\!\!\!\perp Y_i | \mathcal{V}_i\}$.

Each such set $\mathcal{C}$, defines a family of distributions that satisfy all the statements in $\mathcal{C}$:

$$\mathcal{P}_{\mathcal{C}(G)} = \{P(\mathcal{X}) \ : \ P(X_i, Y_i|\mathcal{V}_i) = P(X_i|\mathcal{V}_i)P(Y_i|\mathcal{V}_i) \text{ for all } X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i \text{ in } \mathcal{C} \}$$

G may also encode a family of distributions by their functional form, e.g. for a factor graph

$$\mathcal{P}_G = \{P(\mathcal{X}) \ : \ P(\mathcal{X}) = \tfrac{1}{Z}\prod_j f_j(\mathcal{X}_{C_j}), \text{ for some non-negative functions } f_j \}$$

- For directed graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$.
- For undirected graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$ if all distributions are positive, i.e. $P(\mathcal{X}) > 0$ for all values of $\mathcal{X}$ (Hammersley-Clifford Theorem).

Graphs, conditional independencies, and families of distributions

Each graph G implies a set of conditional independence statements $\mathcal{C}(G) = \{X_i \perp\!\!\!\perp Y_i | \mathcal{V}_i\}$.

Each such set $\mathcal{C}$, defines a family of distributions that satisfy all the statements in $\mathcal{C}$:

$$\mathcal{P}_{\mathcal{C}(G)} = \{P(\mathcal{X}) \ : \ P(X_i, Y_i|\mathcal{V}_i) = P(X_i|\mathcal{V}_i)P(Y_i|\mathcal{V}_i) \text{ for all } X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i \text{ in } \mathcal{C} \ \}$$

G may also encode a family of distributions by their functional form, e.g. for a factor graph

$$\mathcal{P}_G = \{P(\mathcal{X}) \ : \ P(\mathcal{X}) = \tfrac{1}{Z}\textstyle\prod_j f_j(\mathcal{X}_{C_j}), \text{ for some non-negative functions } f_j \ \}$$

- For directed graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$.
- For undirected graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$ if all distributions are positive, i.e. $P(\mathcal{X}) > 0$ for all values of $\mathcal{X}$ (Hammersley-Clifford Theorem).
- There are factor graphs for which $\mathcal{P}_G \neq \mathcal{P}_{\mathcal{C}_G}$.

Graphs, conditional independencies, and families of distributions

Each graph G implies a set of conditional independence statements $\mathcal{C}(G) = \{X_i \perp\!\!\!\perp Y_i | \mathcal{V}_i\}$.

Each such set $\mathcal{C}$, defines a family of distributions that satisfy all the statements in $\mathcal{C}$:

$$\mathcal{P}_{\mathcal{C}(G)} = \{P(\mathcal{X}) \ : \ P(X_i, Y_i|\mathcal{V}_i) = P(X_i|\mathcal{V}_i)P(Y_i|\mathcal{V}_i) \text{ for all } X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i \text{ in } \mathcal{C} \ \}$$

G may also encode a family of distributions by their functional form, e.g. for a factor graph

$$\mathcal{P}_G = \{P(\mathcal{X}) \ : \ P(\mathcal{X}) = \tfrac{1}{Z}\textstyle\prod_j f_j(\mathcal{X}_{C_j}), \text{ for some non-negative functions } f_j \ \}$$

- For directed graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$.
- For undirected graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$ if all distributions are positive, i.e. $P(\mathcal{X}) > 0$ for all values of $\mathcal{X}$ (Hammersley-Clifford Theorem).
- There are factor graphs for which $\mathcal{P}_G \neq \mathcal{P}_{\mathcal{C}_G}$.
- Factor graphs are more expressive than undirected graphs: for every undirected graph G_1 there is a factor graph G_2 with $\mathcal{P}_{G_1} = \mathcal{P}_{G_2}$ but not vice versa.

Graphs, conditional independencies, and families of distributions

Each graph G implies a set of conditional independence statements $\mathcal{C}(G) = \{X_i \perp\!\!\!\perp Y_i | \mathcal{V}_i\}$.

Each such set $\mathcal{C}$, defines a family of distributions that satisfy all the statements in $\mathcal{C}$:

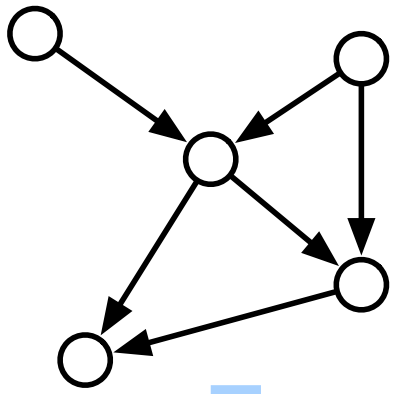
$$\mathcal{P}_{\mathcal{C}(G)} = \{P(\mathcal{X}) \ : \ P(X_i, Y_i|\mathcal{V}_i) = P(X_i|\mathcal{V}_i)P(Y_i|\mathcal{V}_i) \text{ for all } X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i \text{ in } \mathcal{C} \ \}$$

G may also encode a family of distributions by their functional form, e.g. for a factor graph

$$\mathcal{P}_G = \{P(\mathcal{X}) \ : \ P(\mathcal{X}) = \tfrac{1}{Z}\textstyle\prod_j f_j(\mathcal{X}_{C_j}), \text{ for some non-negative functions } f_j \ \}$$

- For directed graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$.
- For undirected graphs, $\mathcal{P}_G = \mathcal{P}_{\mathcal{C}(G)}$ if all distributions are positive, i.e. $P(\mathcal{X}) > 0$ for all values of $\mathcal{X}$ (Hammersley-Clifford Theorem).
- There are factor graphs for which $\mathcal{P}_G \neq \mathcal{P}_{\mathcal{C}_G}$.
- Factor graphs are more expressive than undirected graphs: for every undirected graph G_1 there is a factor graph G_2 with $\mathcal{P}_{G_1} = \mathcal{P}_{G_2}$ but not vice versa.
- Adding edges to graph $\Rightarrow$ removing conditional independency statements $\Rightarrow$ enlarging the family of distributions (converse true for removing edges).

Graphs, conditional independencies, and families of distributions

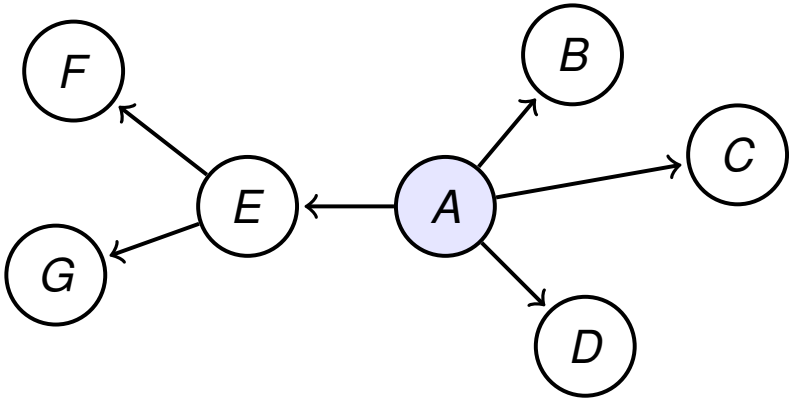


$$\{X_i \perp\!\!\!\perp Y_i | \mathcal{V}_i\}$$

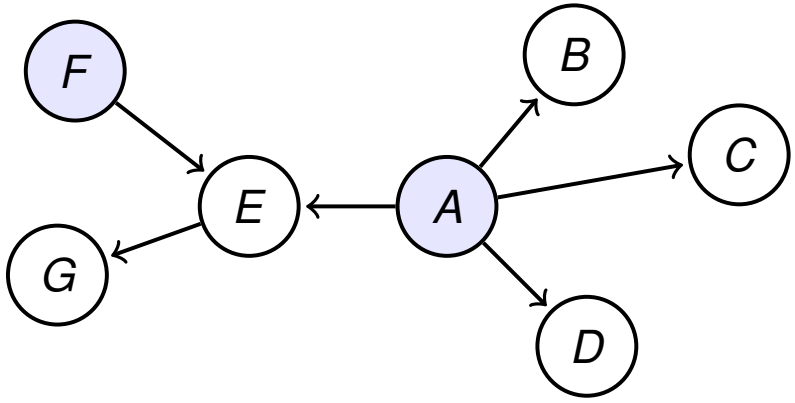
$$\{p(\mathbf{X}) = \textstyle\prod_i p(X_i | X_{\mathsf{pa}(i)})\}$$

$$\{p(\mathbf{X}) : p(X_i, Y_i | \mathcal{V}_i) = p(X_i | \mathcal{V}_i) p(Y_i | \mathcal{V}_i)\}$$

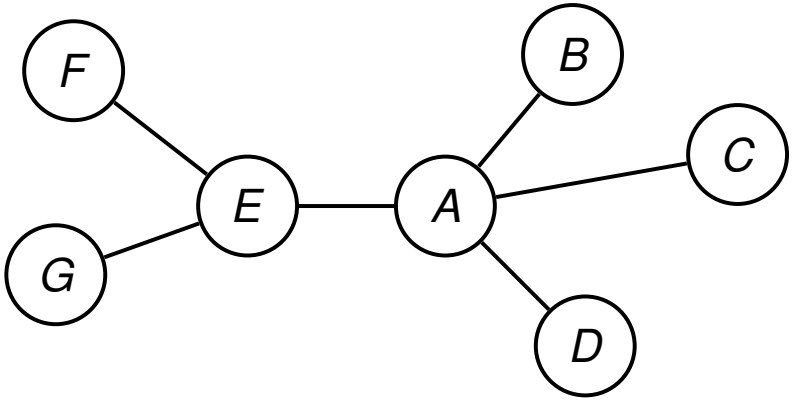
Tree-structured graphical models



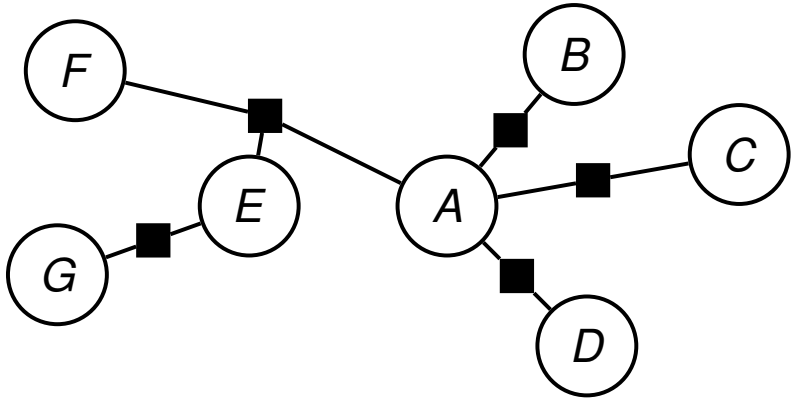
Rooted directed tree



Directed polytree



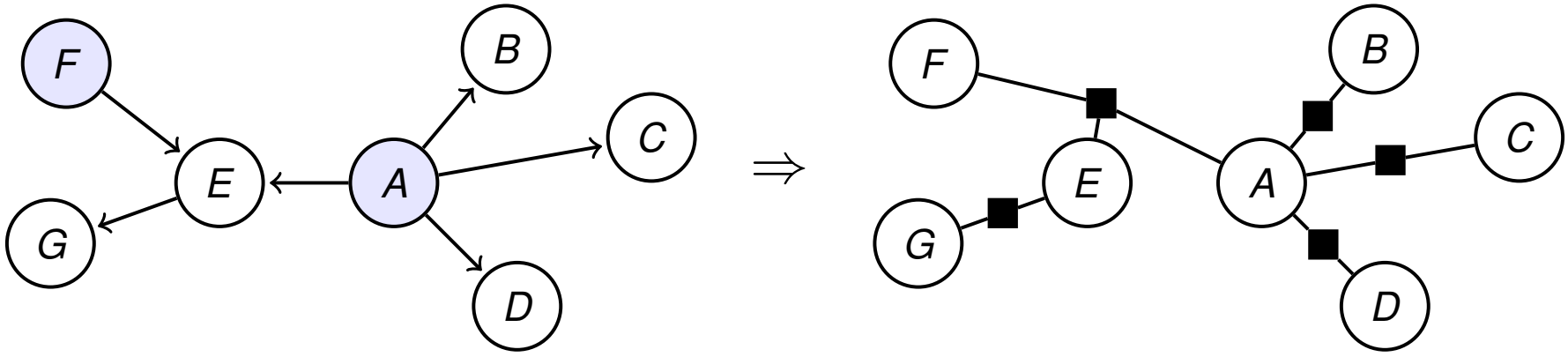
Undirected tree



Tree-structured factor graph

These are all tree-structured or “singly-connected” graphs.

Polytrees to tree-structured factor graphs



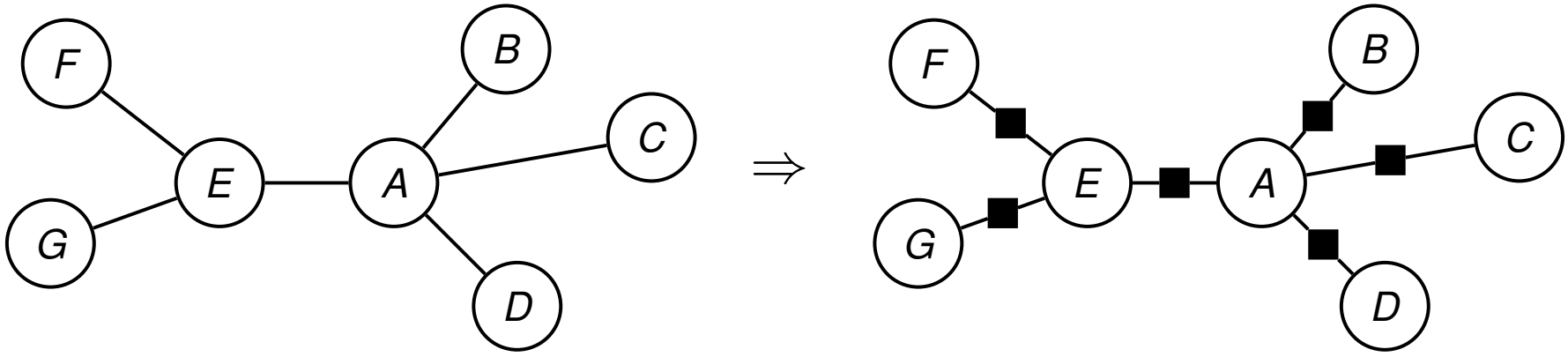
Polytrees are tree-structured DAGs that may have more than one root.

$$P(\mathcal{X}) = \prod_i P(X_i|X_{\text{pa}(i)})$$
$$= \prod_i f_i(X_{C_i})$$

where $C_i = i \cup \text{pa}(i)$ and $f_i(X_{C_i}) = P(X_i|X_{\text{pa}(i)})$.

Marginal distribution on roots $P(X_r)$ absorbed into an adjacent factor.

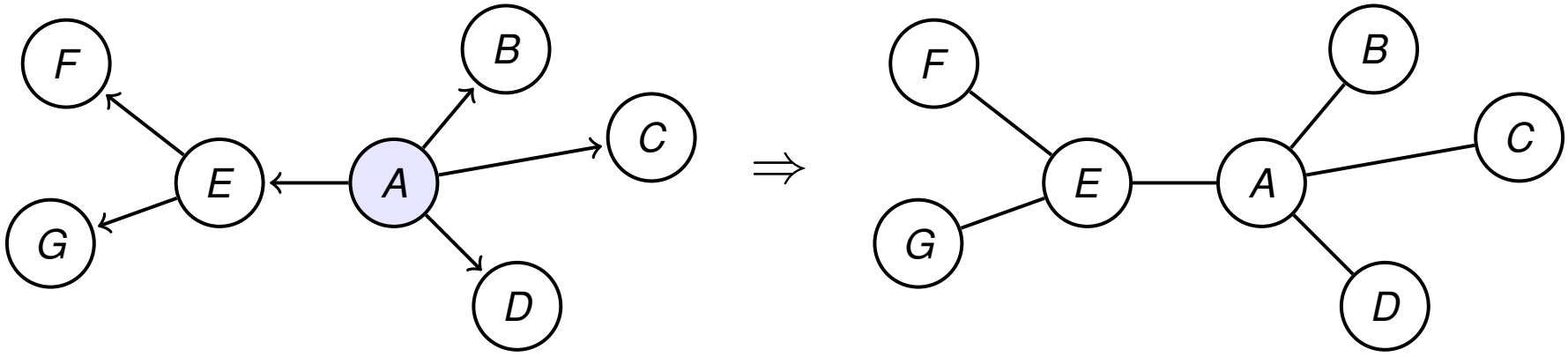
Undirected trees and factor graphs



In an undirected tree all maximal cliques are of size 2, and so the equivalent factor graph has only pairwise factors.

$$P(\mathcal{X}) = \frac{1}{Z} \prod_{\text{edges } (ij)} f_{(ij)}(X_i, X_j)$$

Rooted directed trees to undirected trees

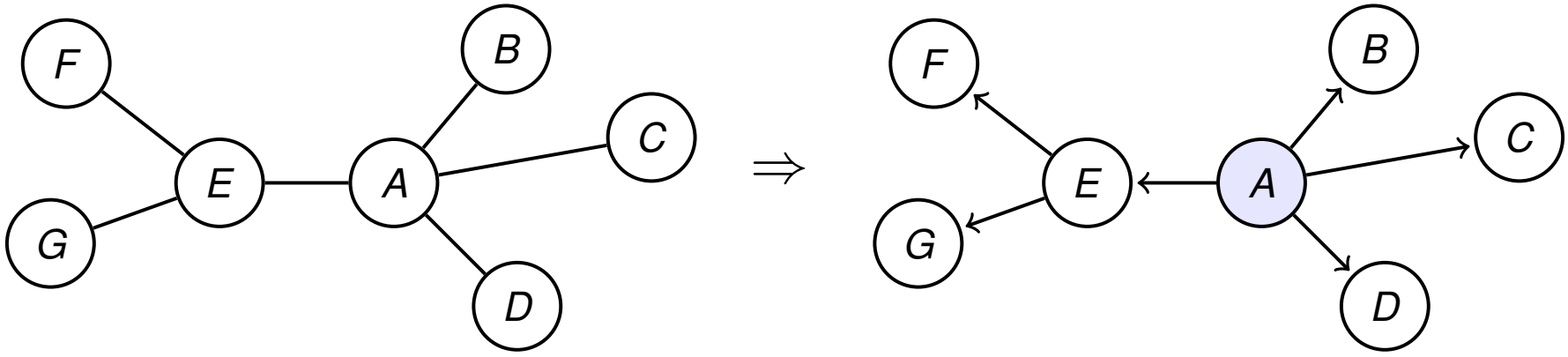


The distribution for a single-rooted directed tree can be written as a product of pairwise factors $\Rightarrow$ undirected tree.

$$P(\mathcal{X}) = P(X_r) \prod_{i \neq r} P(X_i | X_{\text{pa}(i)})$$

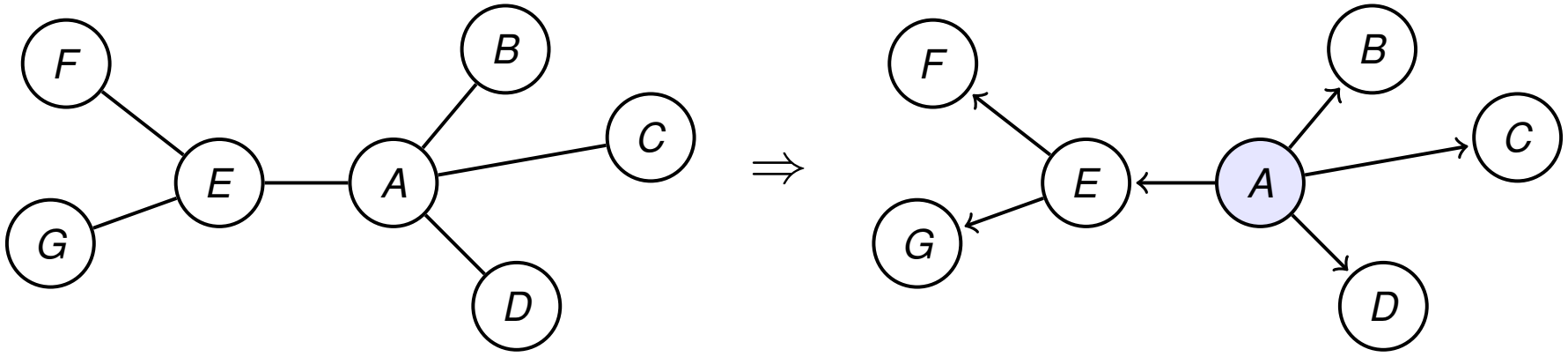
$$= \prod_{\text{edges } (ij)} f_{(ij)}(X_i, X_j)$$

Undirected trees to rooted directed trees



This direction is slightly trickier:

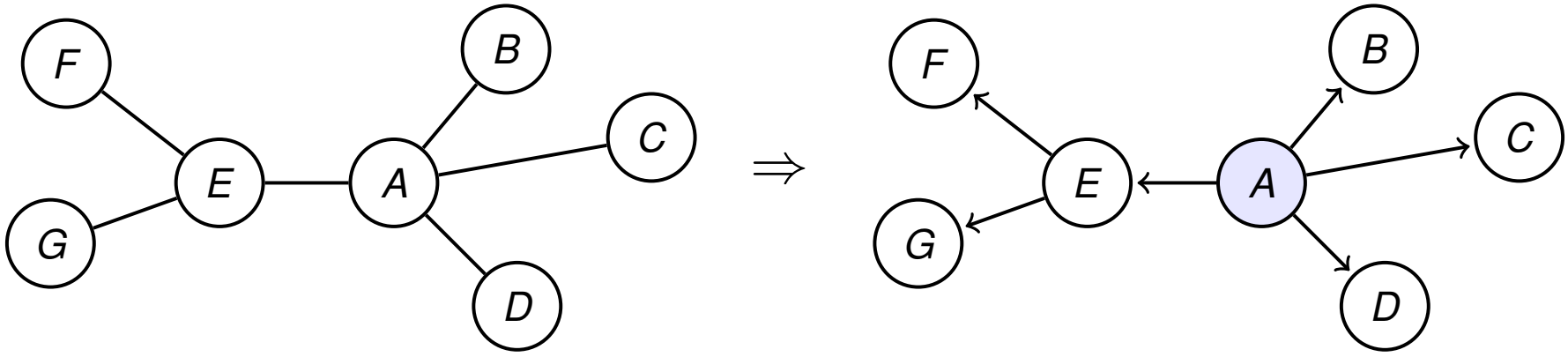
Undirected trees to rooted directed trees



This direction is slightly trickier:

- Choose an arbitrary node X_r to be the root and point all the arrows away from it

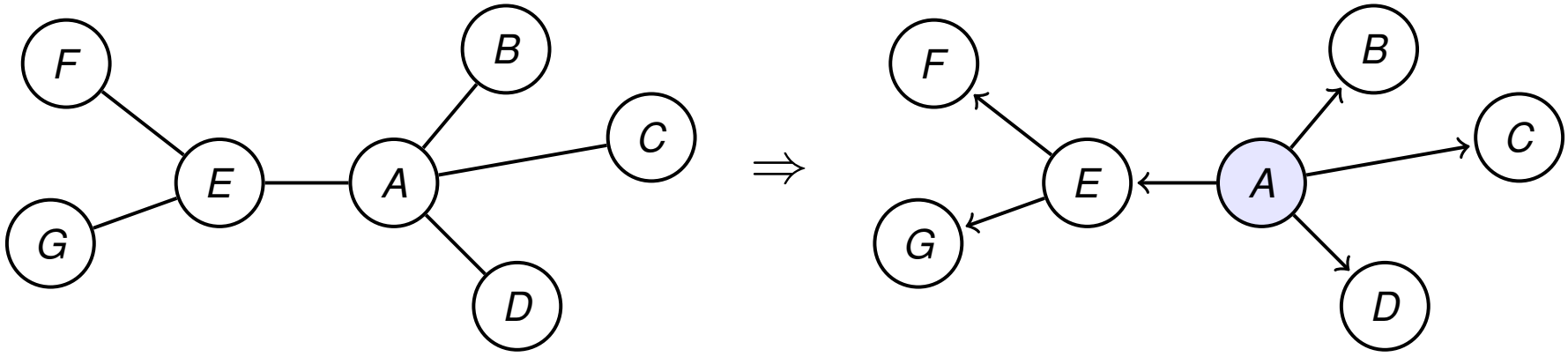
Undirected trees to rooted directed trees



This direction is slightly trickier:

- Choose an arbitrary node X_r to be the root and point all the arrows away from it
- Compute the marginal distributions on single nodes $P(X_i)$ and on edges $P(X_i, X_j)$ implied by the undirected graph.

Undirected trees to rooted directed trees



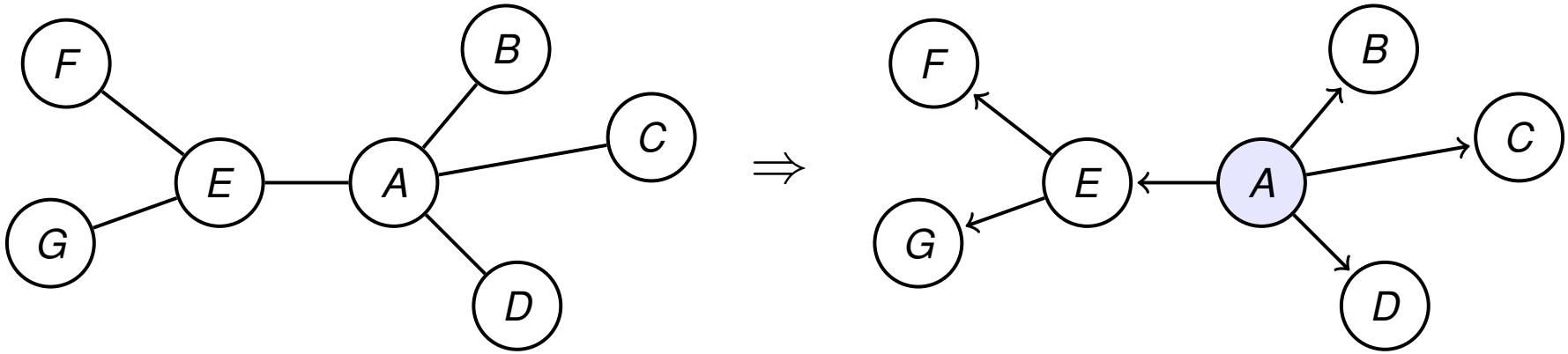
This direction is slightly trickier:

- Choose an arbitrary node X_r to be the root and point all the arrows away from it
- Compute the marginal distributions on single nodes $P(X_i)$ and on edges $P(X_i, X_j)$ implied by the undirected graph.
- Compute the conditionals in the DAG:

$$P(\mathcal{X}) = P(X_r) \prod_{i \neq r} P(X_i | X_{\text{pa}(i)}) = P(X_r) \prod_{i \neq r} \frac{P(X_i, X_{\text{pa}(i)})}{P(X_{\text{pa}(i)})}$$

$$= \frac{\prod_{\text{edges } (ij)} P(X_i, X_j)}{\prod_{\text{nodes } i} P(X_i)^{\text{deg(i)-1}}}$$

Undirected trees to rooted directed trees



This direction is slightly trickier:

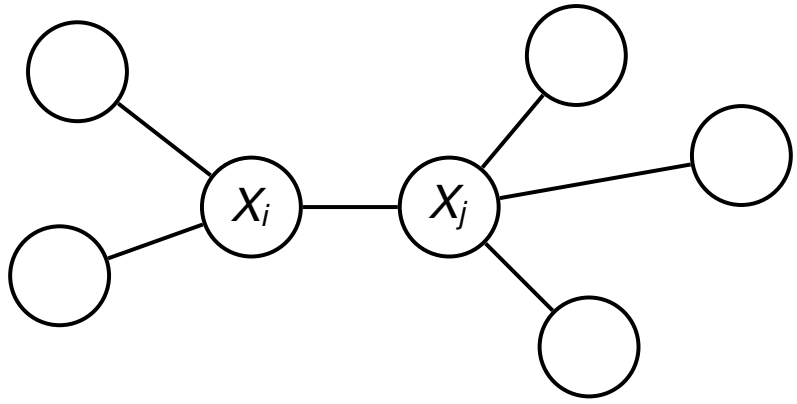
- Choose an arbitrary node X_r to be the root and point all the arrows away from it
- Compute the marginal distributions on single nodes $P(X_i)$ and on edges $P(X_i, X_j)$ implied by the undirected graph.
- Compute the conditionals in the DAG:

$$P(\mathcal{X}) = P(X_r) \prod_{i \neq r} P(X_i | X_{\text{pa}(i)}) = P(X_r) \prod_{i \neq r} \frac{P(X_i, X_{\text{pa}(i)})}{P(X_{\text{pa}(i)})}$$

$$= \frac{\prod_{\text{edges } (ij)} P(X_i, X_j)}{\prod_{\text{nodes } i} P(X_i)^{\text{deg(i)-1}}}$$

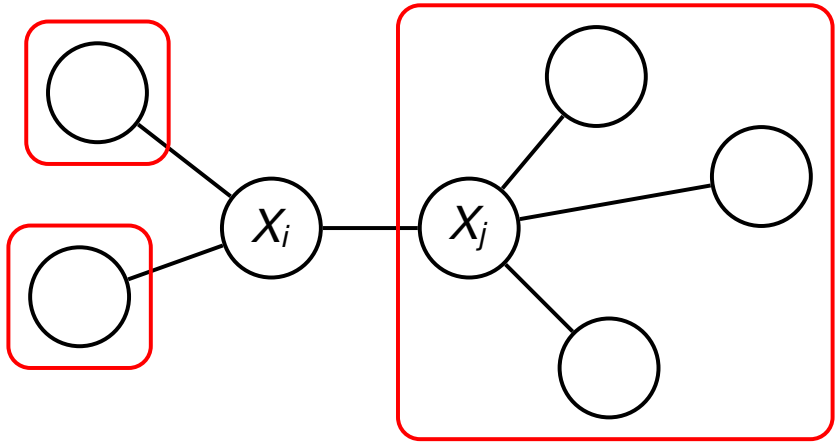
How do we compute $P(X_i)$ and $P(X_i, X_j)$? $\Rightarrow$ **Belief propagation**.

Finding marginals in undirected trees



Undirected tree $\Rightarrow$ pairwise factored joint distribution: $P(\mathcal{X}) = \frac{1}{Z} \prod_{(ij) \in \mathcal{E}_T} f_{(ij)}(X_i, X_j)$

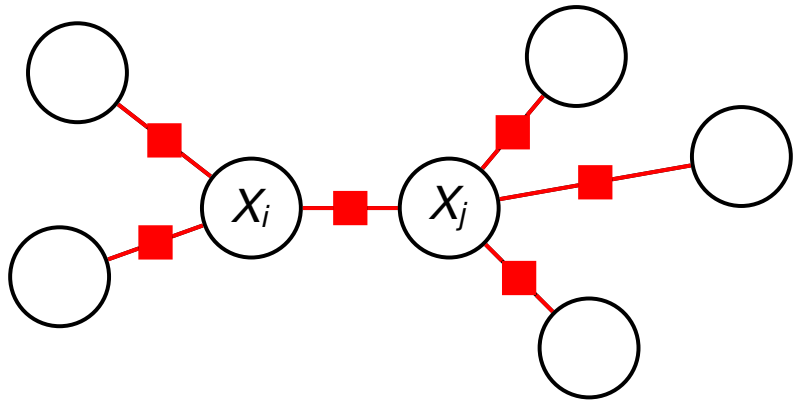
Finding marginals in undirected trees



Undirected tree $\Rightarrow$ pairwise factored joint distribution: $P(\mathcal{X}) = \frac{1}{Z} \prod_{(ij) \in \mathcal{E}_T} f_{(ij)}(X_i, X_j)$

Each neigbour X_j of X_i defines a disjoint subtree $T_{j \to i}$.

Finding marginals in undirected trees

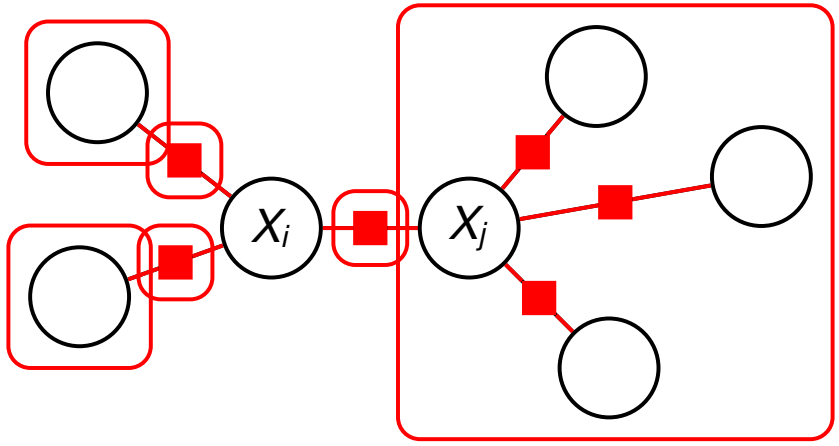


Undirected tree $\Rightarrow$ pairwise factored joint distribution: $P(\mathcal{X}) = \frac{1}{Z} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$

Each neigbour X_j of X_i defines a disjoint subtree $T_{j\to i}$. So we can split up the product:

$$P(X_i) = \sum_{\mathcal{X}\setminus\{X_i\}} P(\mathcal{X}) \propto \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$$

Finding marginals in undirected trees



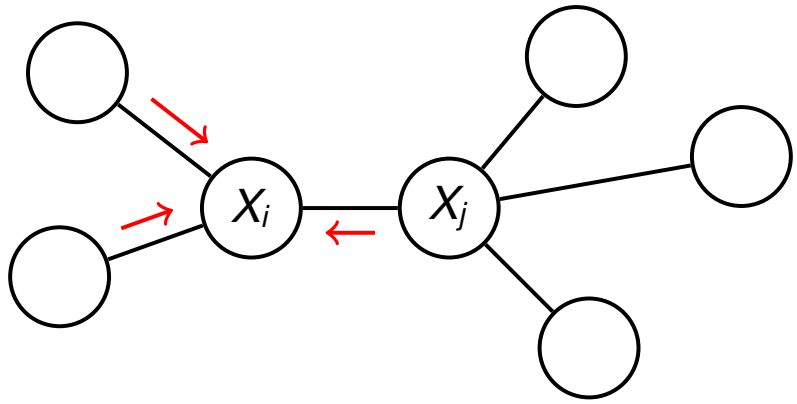
Undirected tree $\Rightarrow$ pairwise factored joint distribution: $P(\mathcal{X}) = \frac{1}{Z} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$

Each neigbour X_j of X_i defines a disjoint subtree $T_{j\to i}$. So we can split up the product:

$$P(X_i) = \sum_{\mathcal{X}\setminus\{X_i\}} P(\mathcal{X}) \propto \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$$

$$= \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{X_j\in \mathrm{ne}(X_i)} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

Finding marginals in undirected trees

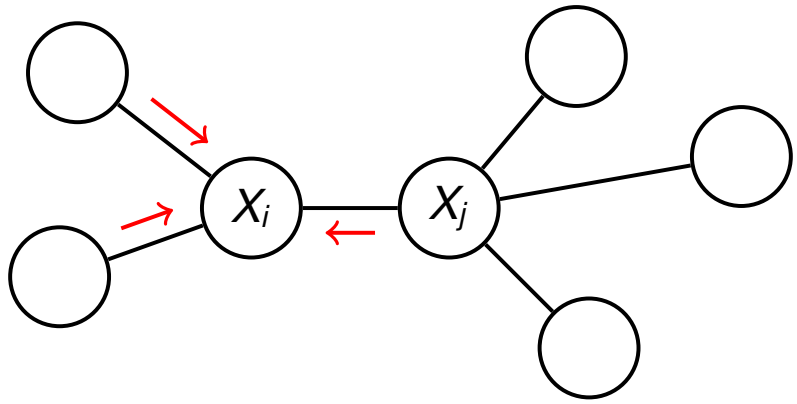


Undirected tree $\Rightarrow$ pairwise factored joint distribution: $P(\mathcal{X}) = \frac{1}{Z} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$

Each neigbour X_j of X_i defines a disjoint subtree $T_{j\to i}$. So we can split up the product:

$$
\begin{aligned}
P(X_i) = \sum_{\mathcal{X}\setminus\{X_i\}} P(\mathcal{X}) &\propto \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j) \\
&= \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{X_j\in\mathrm{ne}(X_i)} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \\
&= \prod_{X_j\in\mathrm{ne}(X_i)} \underbrace{\left(\sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \right)}_{M_{j\to i}(X_i)}
\end{aligned}
$$

Finding marginals in undirected trees

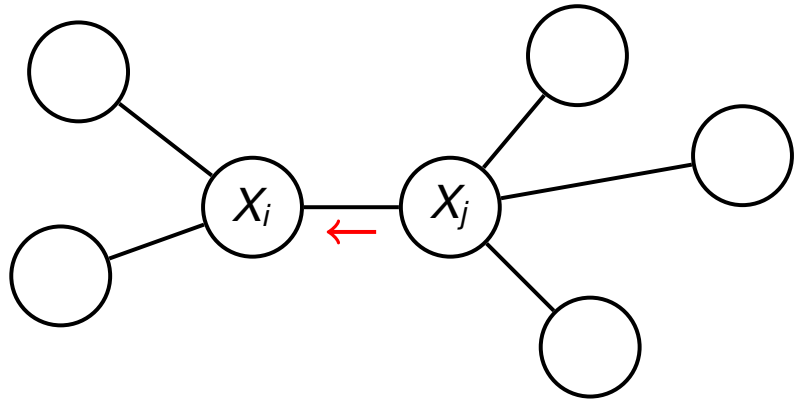


Undirected tree $\Rightarrow$ pairwise factored joint distribution: $P(\mathcal{X}) = \frac{1}{Z} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$

Each neigbour X_j of X_i defines a disjoint subtree $T_{j\to i}$. So we can split up the product:

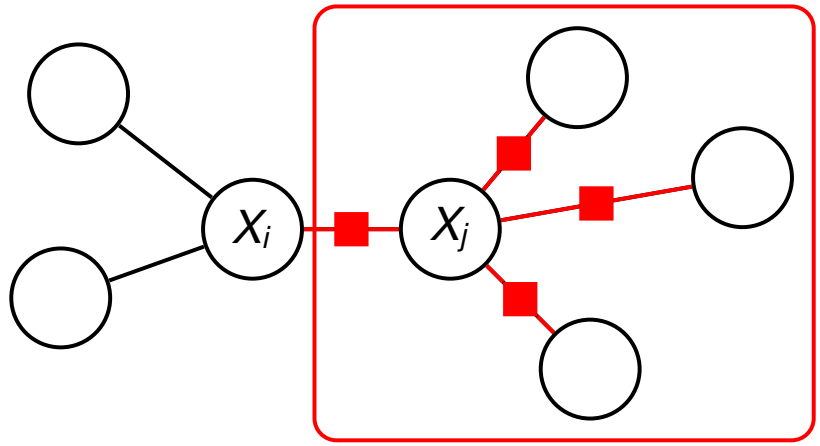
$$
\begin{aligned}
P(X_i) = \sum_{\mathcal{X}\setminus\{X_i\}} P(\mathcal{X}) &\propto \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j) \\
&= \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{X_j\in\mathrm{ne}(X_i)} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \\
&= \prod_{X_j\in\mathrm{ne}(X_i)} \underbrace{\left(\sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \right)}_{M_{j\to i}(X_i)} = \prod_{X_j\in\mathrm{ne}(X_i)} M_{j\to i}(X_i)
\end{aligned}
$$

Message recursion: Belief Propagation (BP)



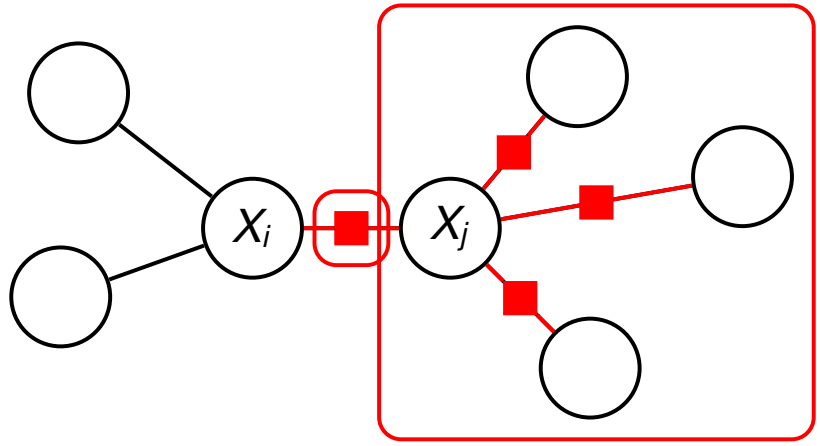
$M_{j \to i}(X_i)$

Message recursion: Belief Propagation (BP)



$$M_{j\to i}(X_i) = \sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

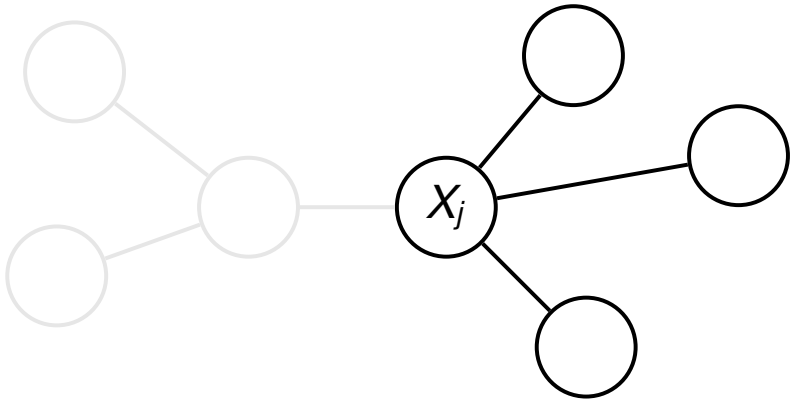
Message recursion: Belief Propagation (BP)



$$M_{j\to i}(X_i) = \sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

$$= \sum_{X_j} f_{(ij)}(X_i, X_j) \sum_{\mathcal{X}_{T_{j\to i}}\setminus X_j} \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

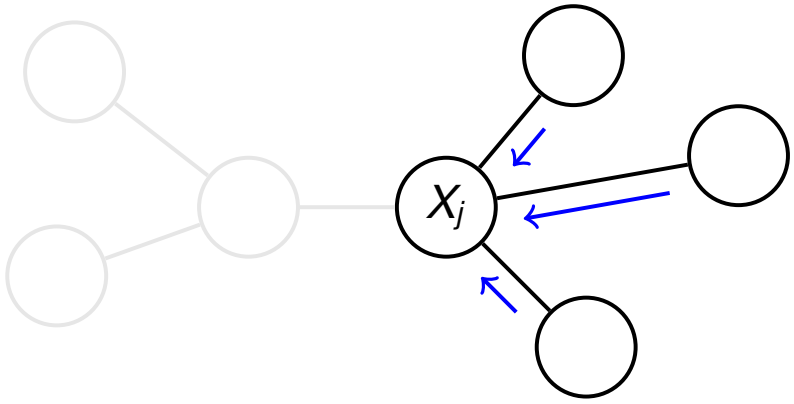
Message recursion: Belief Propagation (BP)



$$M_{j\to i}(X_i) = \sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

$$= \sum_{X_j} f_{(ij)}(X_i, X_j) \underbrace{\sum_{\mathcal{X}_{T_{j\to i}}\setminus X_j} \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})}_{\propto P_{T_{j\to i}}(X_j)}$$

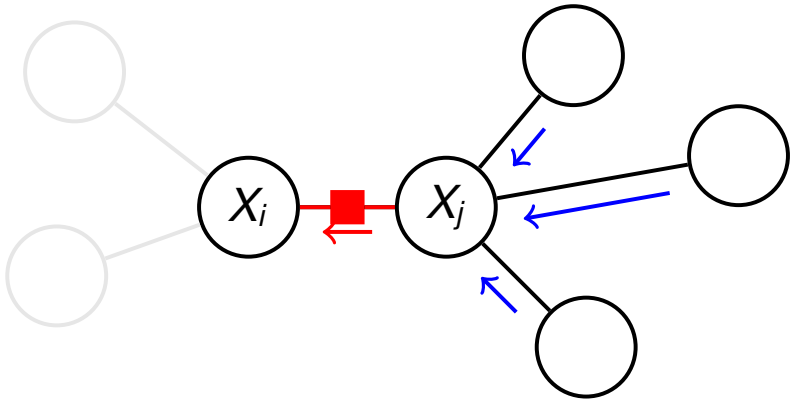
Message recursion: Belief Propagation (BP)



$$M_{j\to i}(X_i) = \sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

$$= \sum_{X_j} f_{(ij)}(X_i, X_j) \underbrace{\sum_{\mathcal{X}_{T_{j\to i}}\setminus X_j} \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})}_{\propto P_{T_{j\to i}}(X_j) \propto \prod_{X_k\in \mathrm{ne}(X_j)\setminus X_i} M_{k\to j}(X_j)}$$

Message recursion: Belief Propagation (BP)

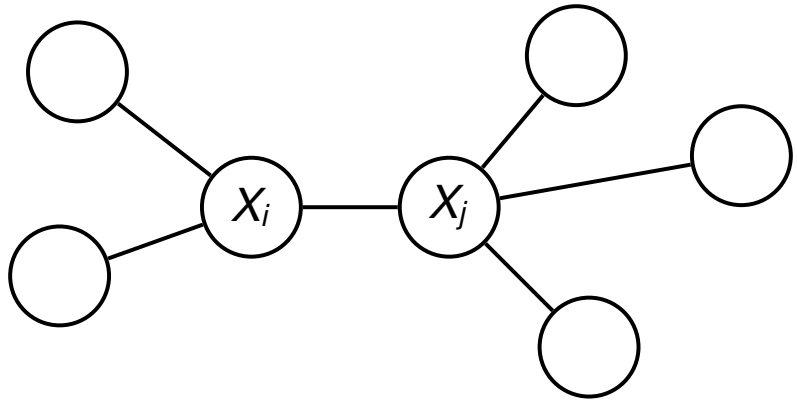


$$M_{j\to i}(X_i) = \sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

$$= \sum_{X_j} f_{(ij)}(X_i, X_j) \underbrace{\sum_{\mathcal{X}_{T_{j\to i}}\setminus X_j} \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})}_{\propto P_{T_{j\to i}}(X_j) \propto \prod_{X_k\in \mathrm{ne}(X_j)\setminus X_i} M_{k\to j}(X_j)}$$

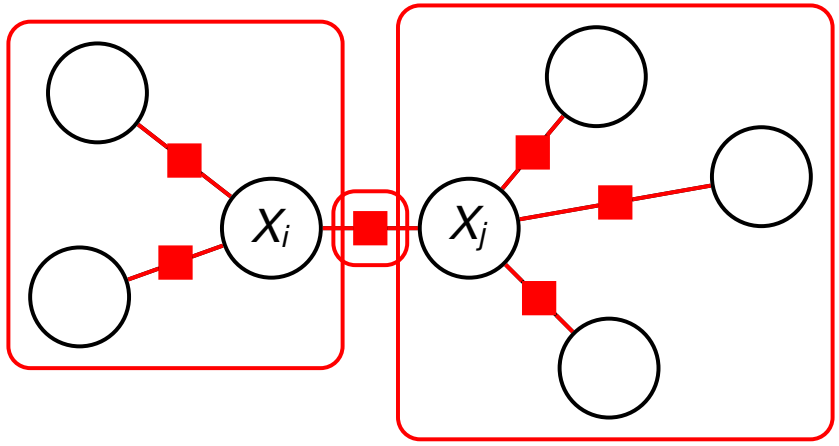
$$= \sum_{X_j} f_{(ij)}(X_i, X_j) \prod_{X_k\in \mathrm{ne}(X_j)\setminus X_i} M_{k\to j}(X_j)$$

BP for pairwise marginals in undirected trees



$$P(X_i, X_j) = \sum_{\mathcal{X} \setminus \{X_i, X_j\}} P(\mathcal{X}) \propto \sum_{\mathcal{X} \setminus \{X_i\}} \prod_{(ij) \in \mathcal{E}_T} f_{(ij)}(X_i, X_j)$$

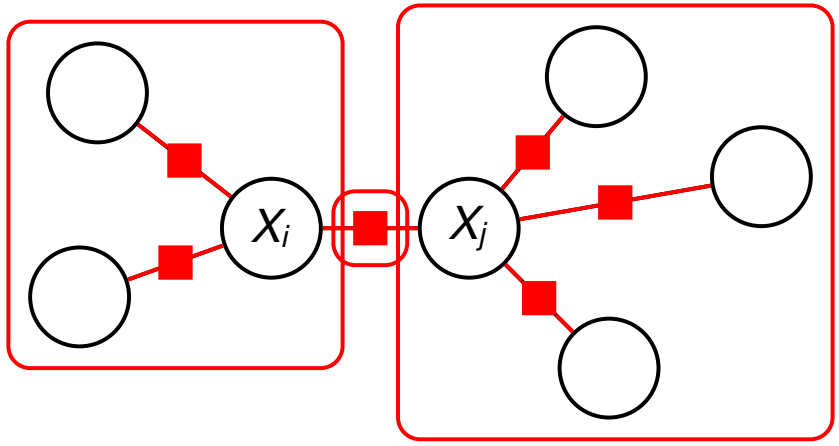
BP for pairwise marginals in undirected trees



$$P(X_i, X_j) = \sum_{\mathcal{X}\setminus\{X_i,X_j\}} P(\mathcal{X}) \propto \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$$

$$= \sum_{\mathcal{X}\setminus\{X_i,X_j\}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \prod_{(i'j')\in\mathcal{E}_{T_{i\to j}}} f_{(i'j')}(X_{i'}, X_{j'})$$

BP for pairwise marginals in undirected trees

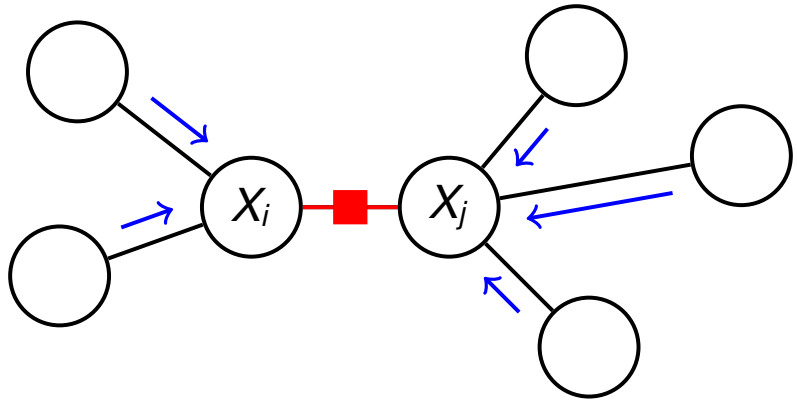


$$P(X_i, X_j) = \sum_{\mathcal{X}\setminus\{X_i,X_j\}} P(\mathcal{X}) \propto \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j)$$

$$= \sum_{\mathcal{X}\setminus\{X_i,X_j\}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \prod_{(i'j')\in\mathcal{E}_{T_{i\to j}}} f_{(i'j')}(X_{i'}, X_{j'})$$

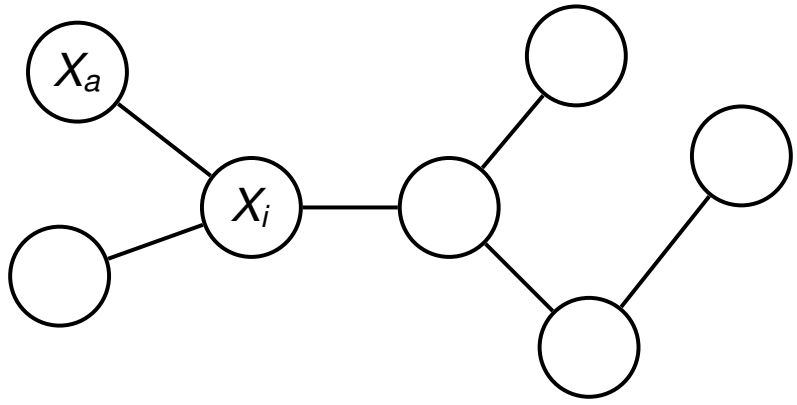
$$= f_{(ij)}(X_i, X_j) \Big(\sum_{\mathcal{X}_{T_{j\to i}}\setminus X_j} \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \Big) \Big(\sum_{\mathcal{X}_{T_{i\to j}}\setminus X_i} \prod_{(i'j')\in\mathcal{E}_{T_{i\to j}}} f_{(i'j')}(X_{i'}, X_{j'}) \Big)$$

BP for pairwise marginals in undirected trees



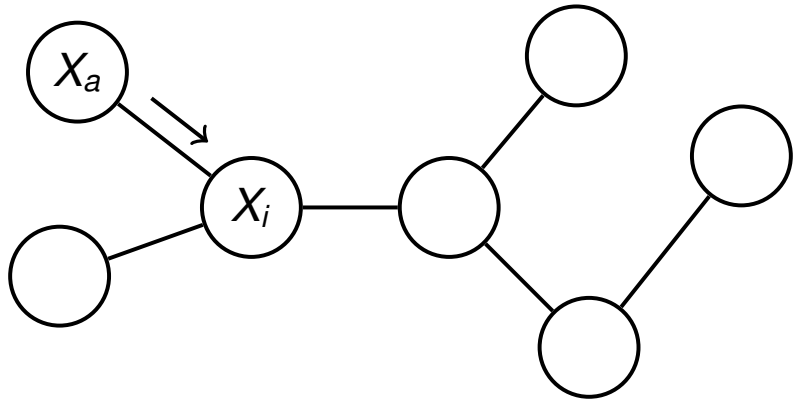
$$
\begin{aligned}
P(X_i, X_j) &= \sum_{\mathcal{X}\setminus\{X_i,X_j\}} P(\mathcal{X}) \propto \sum_{\mathcal{X}\setminus\{X_i\}} \prod_{(ij)\in\mathcal{E}_T} f_{(ij)}(X_i, X_j) \\
&= \sum_{\mathcal{X}\setminus\{X_i,X_j\}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \prod_{(i'j')\in\mathcal{E}_{T_{i\to j}}} f_{(i'j')}(X_{i'}, X_{j'}) \\
&= f_{(ij)}(X_i, X_j) \Big(\sum_{\mathcal{X}_{T_{j\to i}}\setminus X_j} \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'}) \Big) \Big(\sum_{\mathcal{X}_{T_{i\to j}}\setminus X_i} \prod_{(i'j')\in\mathcal{E}_{T_{i\to j}}} f_{(i'j')}(X_{i'}, X_{j'}) \Big) \\
&= f_{(ij)}(X_i, X_j) \prod_{X_k\in \mathrm{ne}(X_j)\setminus X_i} M_{k\to j}(X_j) \prod_{X_k\in \mathrm{ne}(X_i)\setminus X_j} M_{k\to i}(X_i)
\end{aligned}
$$

BP for inference



Messages from observed leaf nodes are conditioned rather than marginalised:

BP for inference

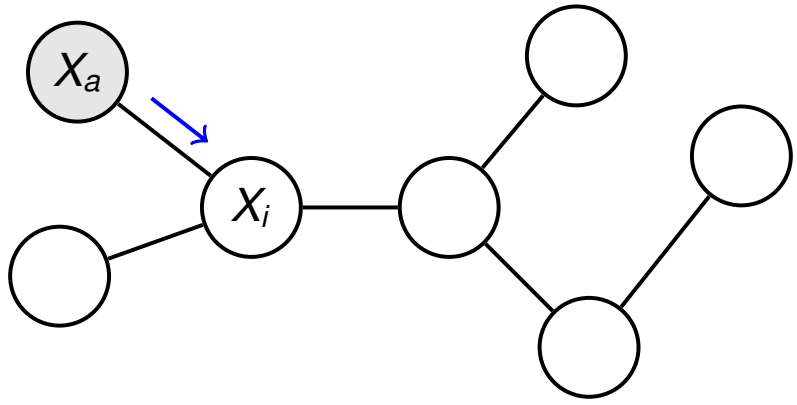


Messages from observed leaf nodes are conditioned rather than marginalised:

To compute $P(X_i)$:

$$M_{a \to i} = \sum_{X_a} f_{ai}(X_a, X_i)$$

BP for inference

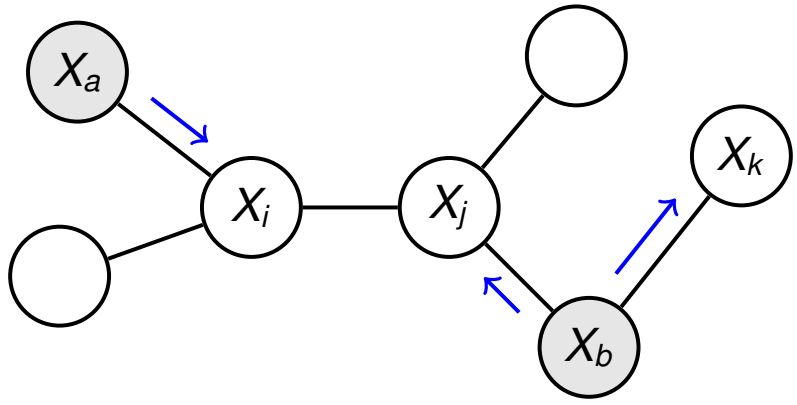


Messages from observed leaf nodes are conditioned rather than marginalised:

To compute $P(X_i)$: $M_{a \to i} = \sum_{X_a} f_{ai}(X_a, X_i)$

To compute $P(X_i | X_a = a)$: $M_{a \to i} = f_{ai}(X_a = a, X_i)$

BP for inference



Messages from observed leaf nodes are conditioned rather than marginalised:

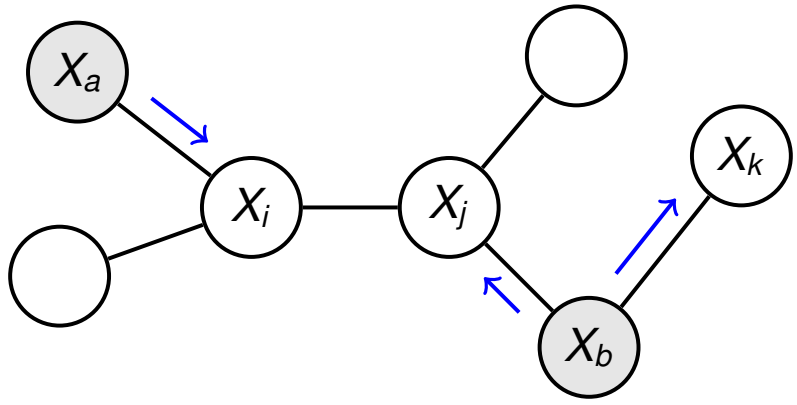
To compute $P(X_i)$: $\quad M_{a\to i} = \sum_{X_a} f_{ai}(X_a, X_i)$

To compute $P(X_i|X_a = a)$: $\quad M_{a\to i} = f_{ai}(X_a = a, X_i)$

Observed internal nodes partition the graph, and so messages propagate independently.

$M_{b\to j} = f_{bj}(X_b = b, X_j)$ $\qquad$ $M_{b\to k} = f_{bk}(X_b = b, X_k)$

BP for inference



Messages from observed leaf nodes are conditioned rather than marginalised:

To compute $P(X_i)$: $\quad M_{a\to i} = \sum_{X_a} f_{ai}(X_a, X_i)$

To compute $P(X_i|X_a = a)$: $\quad M_{a\to i} = f_{ai}(X_a = a, X_i)$

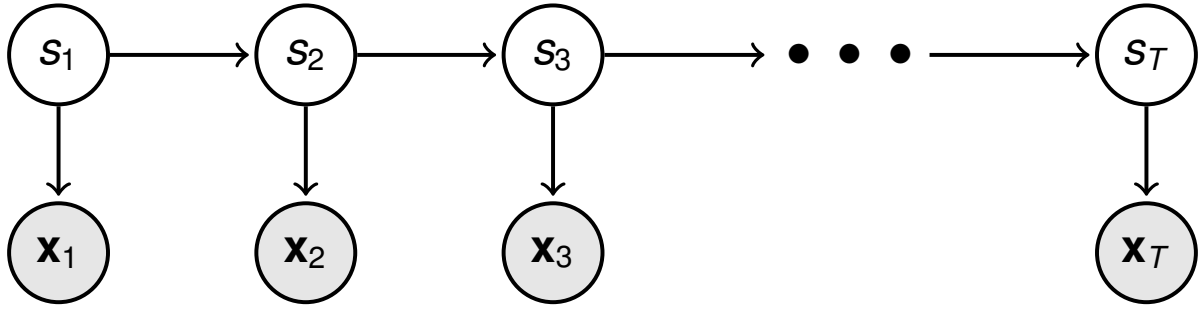
Observed internal nodes partition the graph, and so messages propagate independently.

$M_{b\to j} = f_{bj}(X_b = b, X_j)$ $\qquad$ $M_{b\to k} = f_{bk}(X_b = b, X_k)$

Messages $M_{i\to j}$ are proportional to the likelihood based on any observed variables ($\mathcal{O}$) within the messages subtree $T_{i\to j}$, possibly scaled by a prior factor (depending on factorisation)

$$M_{i\to j}(X_j) \propto P(\mathcal{X}_{T_{i\to j}} \cap \mathcal{O}|X_i)P(X_i)$$

BP for latent chain models



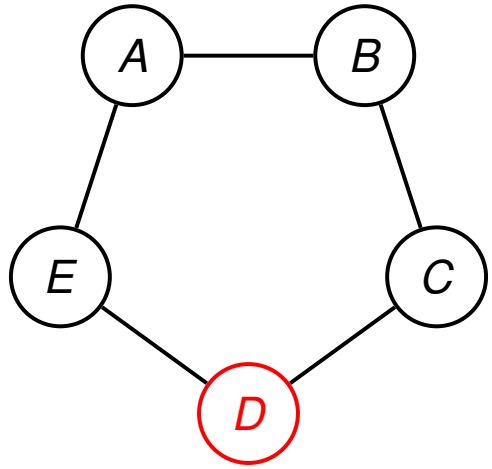
A latent chain model is a rooted directed tree $\Rightarrow$ an undirected tree.

The forward-backward algorithm is just BP on this graph.

$$\alpha_t(i) \Leftrightarrow M_{s_{t-1} \to s_t}(s_t{=}i) \propto P(\mathbf{x}_{1:t}, s_t)$$
$$\beta_t(i) \Leftrightarrow M_{s_{t+1} \to s_t}(s_t{=}i) \propto P(\mathbf{x}_{t+1:T} | s_t)$$
$$\alpha_t(i)\beta_t(i) = \prod_{j \in \mathrm{ne}(s_t)} M_{j \to s_t}(s_t{=}i) \propto P(s_t{=}i | \mathcal{O})$$

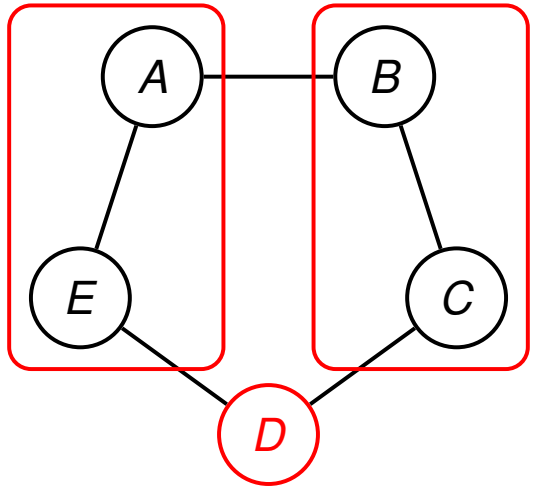
Algorithms like BP extend the power of graphical models beyond just encoding of independence and factorisation. A single derivation serves for a wide array of models.

BP in non-trees?



Can we find $P(D)$ easily?

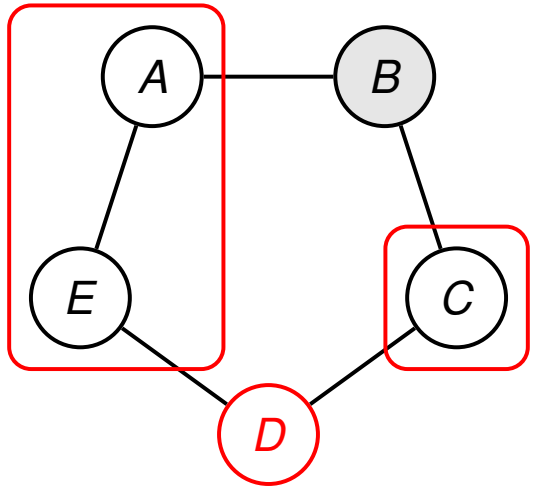
BP in non-trees?



Can we find $P(D)$ easily?

- Neighbours do not belong to disjoint subtrees, so influence of other nodes cannot be separated into messages.

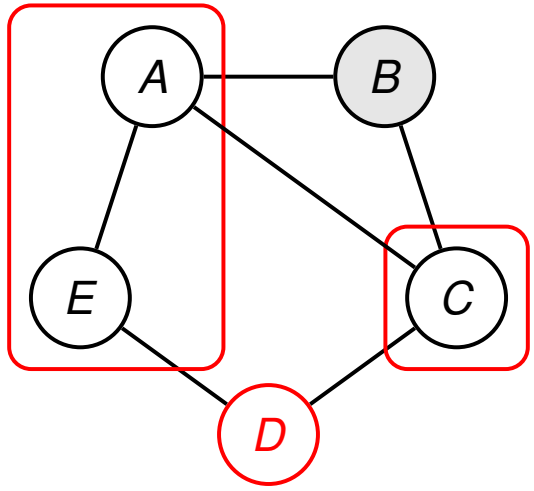
BP in non-trees?



Can we find $P(D)$ easily?

- Neighbours do not belong to disjoint subtrees, so influence of other nodes cannot be separated into messages.
- Observed nodes may break loops and make subtrees independent,

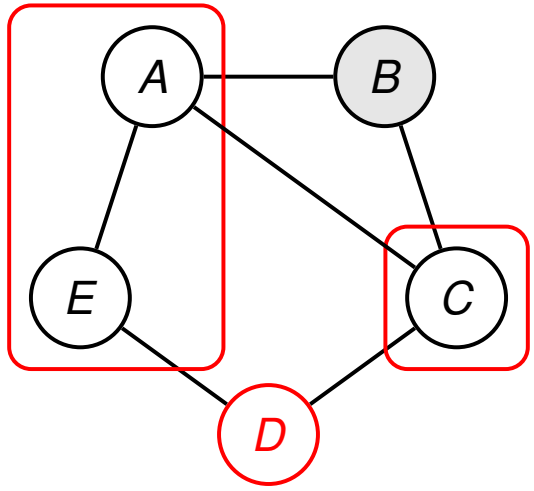
BP in non-trees?



Can we find $P(D)$ easily?

- Neighbours do not belong to disjoint subtrees, so influence of other nodes cannot be separated into messages.
- Observed nodes may break loops and make subtrees independent, but may not resolve all loops.

BP in non-trees?

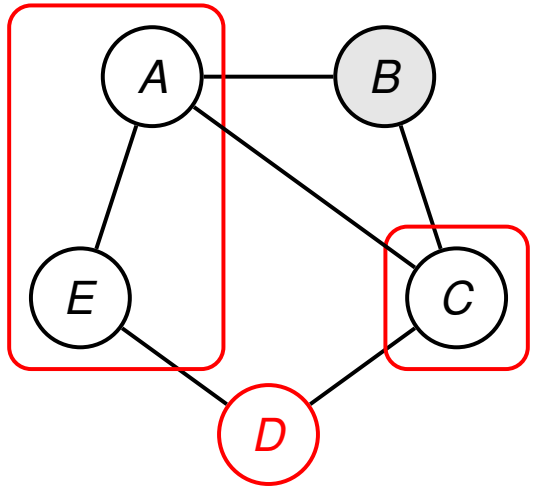


Can we find $P(D)$ easily?

- Neighbours do not belong to disjoint subtrees, so influence of other nodes cannot be separated into messages.
- Observed nodes may break loops and make subtrees independent, but may not resolve all loops.

Possible strategies:

BP in non-trees?



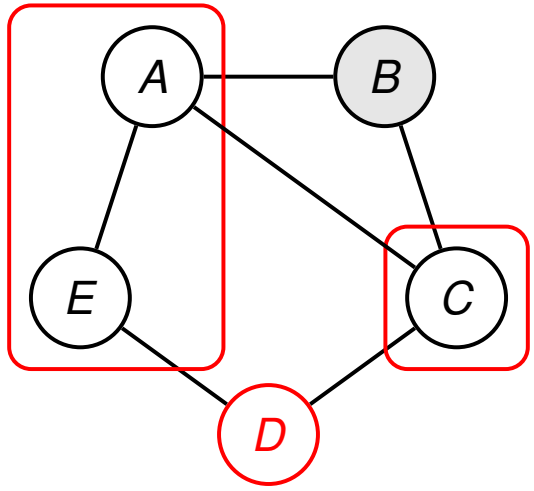
Can we find $P(D)$ easily?

- Neighbours do not belong to disjoint subtrees, so influence of other nodes cannot be separated into messages.
- Observed nodes may break loops and make subtrees independent, but may not resolve all loops.

Possible strategies:

- Propagate local messages anyway, and hope for the best
  - loopy belief propagation — actually an approximation which we will study later.

BP in non-trees?



Can we find $P(D)$ easily?

- Neighbours do not belong to disjoint subtrees, so influence of other nodes cannot be separated into messages.
- Observed nodes may break loops and make subtrees independent, but may not resolve all loops.

Possible strategies:

- Propagate local messages anyway, and hope for the best
  - loopy belief propagation — actually an approximation which we will study later.
- Group variables together into multivariate nodes until the resulting graph is a tree.
  - Junction tree

Graph transformations

For exact inference in arbitrary graphical models we need to transform the given graph into one that will be easier to handle (specifically a tree: the **junction or join tree**).

Graph transformations

For exact inference in arbitrary graphical models we need to transform the given graph into one that will be easier to handle (specifically a tree: the **junction or join tree**).

The original graph G encoded a distribution $P(\mathcal{X})$ with a certain factorisation or independence structure.

Graph transformations

For exact inference in arbitrary graphical models we need to transform the given graph into one that will be easier to handle (specifically a tree: the **junction or join tree**).

The original graph G encoded a distribution $P(\mathcal{X})$ with a certain factorisation or independence structure.

- Transformation from G to an easy-to-handle G' will only be valid if $P(\mathcal{X})$ can also be represented by G'.

Graph transformations

For exact inference in arbitrary graphical models we need to transform the given graph into one that will be easier to handle (specifically a tree: the **junction or join tree**).

The original graph G encoded a distribution $P(\mathcal{X})$ with a certain factorisation or independence structure.

- Transformation from G to an easy-to-handle G' will only be valid if $P(\mathcal{X})$ can also be represented by G'.
- This can be ensured if every step of the graph transformation only removes conditional independencies, never adds them.

Graph transformations

For exact inference in arbitrary graphical models we need to transform the given graph into one that will be easier to handle (specifically a tree: the **junction or join tree**).

The original graph G encoded a distribution $P(\mathcal{X})$ with a certain factorisation or independence structure.

- Transformation from G to an easy-to-handle G' will only be valid if $P(\mathcal{X})$ can also be represented by G'.
- This can be ensured if every step of the graph transformation only removes conditional independencies, never adds them.
- Thus the family of possible encoded distributions grows or stays the same at each step, and $P(\mathcal{X})$ will be in the family of distributions represented by G'.

Graph transformations

For exact inference in arbitrary graphical models we need to transform the given graph into one that will be easier to handle (specifically a tree: the **junction or join tree**).

The original graph G encoded a distribution $P(\mathcal{X})$ with a certain factorisation or independence structure.

- Transformation from G to an easy-to-handle G' will only be valid if $P(\mathcal{X})$ can also be represented by G'.
- This can be ensured if every step of the graph transformation only removes conditional independencies, never adds them.
- Thus the family of possible encoded distributions grows or stays the same at each step, and $P(\mathcal{X})$ will be in the family of distributions represented by G'.
- The factor potentials on the new graph G' are built from those given on G, so as to encode the same distribution.

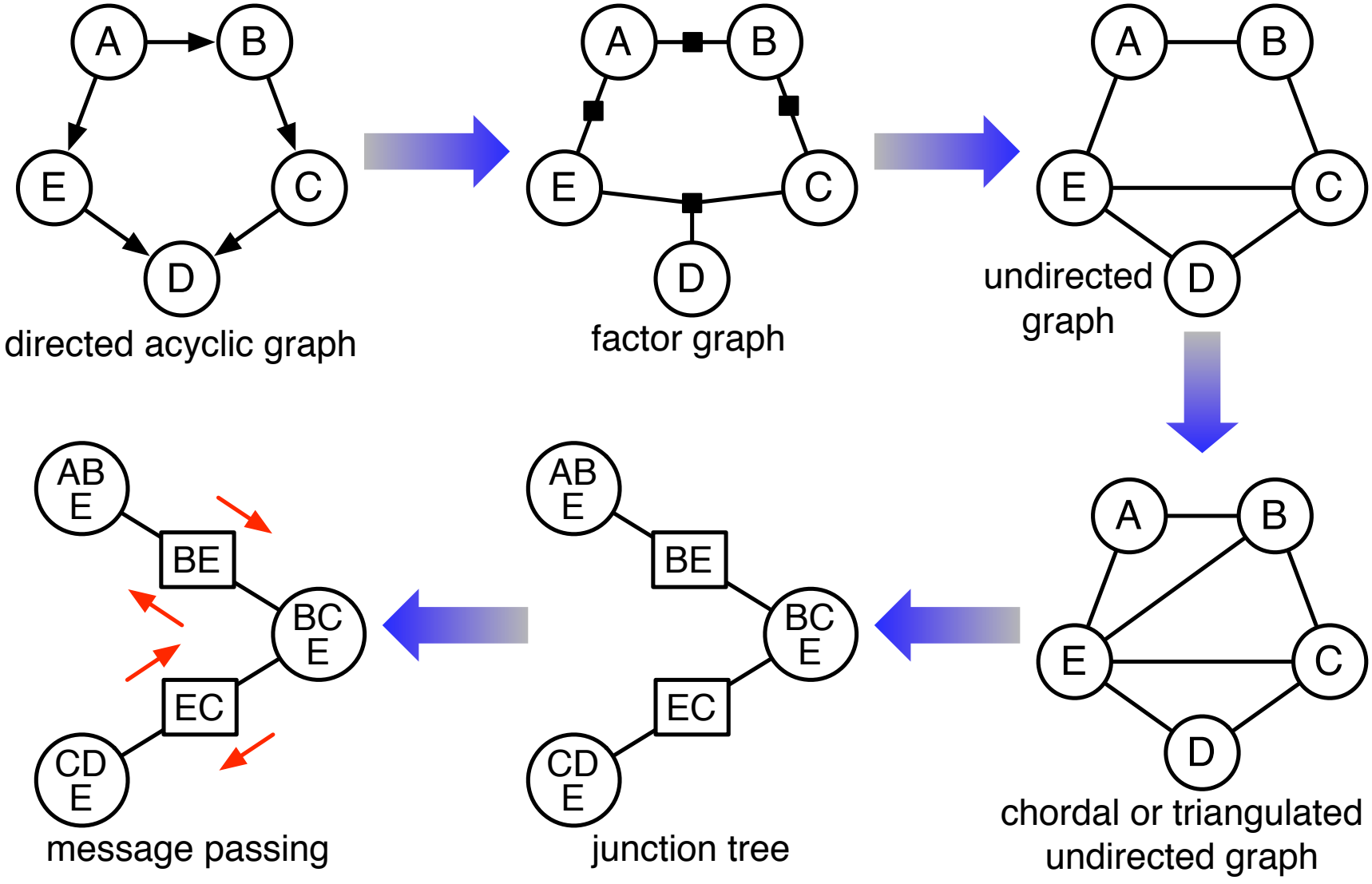
Graph transformations

For exact inference in arbitrary graphical models we need to transform the given graph into one that will be easier to handle (specifically a tree: the **junction or join tree**).

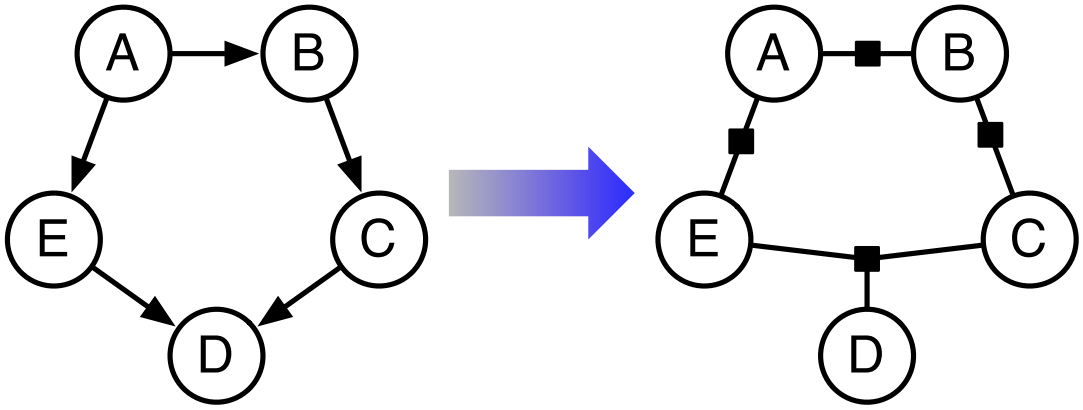
The original graph G encoded a distribution $P(\mathcal{X})$ with a certain factorisation or independence structure.

- Transformation from G to an easy-to-handle G' will only be valid if $P(\mathcal{X})$ can also be represented by G'.
- This can be ensured if every step of the graph transformation only removes conditional independencies, never adds them.
- Thus the family of possible encoded distributions grows or stays the same at each step, and $P(\mathcal{X})$ will be in the family of distributions represented by G'.
- The factor potentials on the new graph G' are built from those given on G, so as to encode the same distribution.
- Then inference on G' with the appropriate potentials acts on $P(\mathcal{X})$.

The Junction Tree algorithm



DAG to factor graph



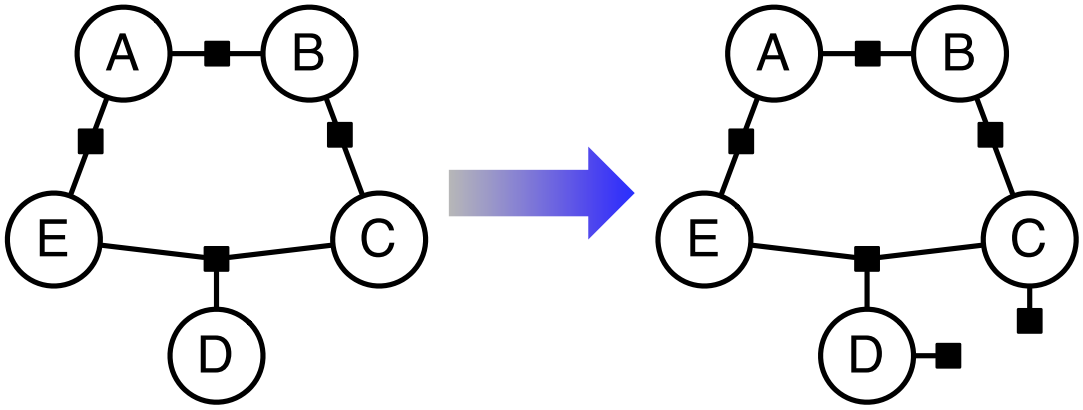
Factors are simply the conditional distributions in the DAG.

$$P(\mathcal{X}) = \prod_i P(X_i | X_{\text{pa}(i)})$$
$$= \prod_i f_i(X_{C_i})$$

where $C_i = i \cup \text{pa}(i)$ and $f_i(X_{C_i}) = P(X_i | X_{\text{pa}(i)})$.

Marginal distribution on root(s) $P(X_r)$ absorbed into an adjacent factor.

Observations in a factor graph



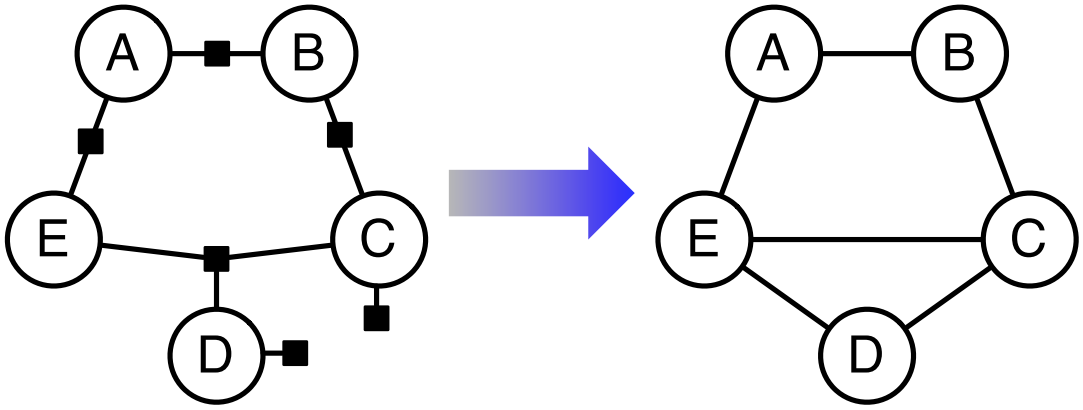
Inference usually targets a posterior marginal given a set of observed values $P(X_I|\mathcal{O})$
e.g. $P(A|D = \text{wet}, C = \text{rain})$.

Formally, we can either modify the factors linked to observed nodes, or add singleton factors adjacent to the observed nodes, e.g.

$$f_D(D) = \begin{cases} 1 & \text{if } D = \text{wet;} \\ 0 & \text{otherwise.} \end{cases}$$

$$f_C(C) = \begin{cases} 1 & \text{if } C = \text{rain;} \\ 0 & \text{otherwise.} \end{cases}$$

Factor graph to undirected graph



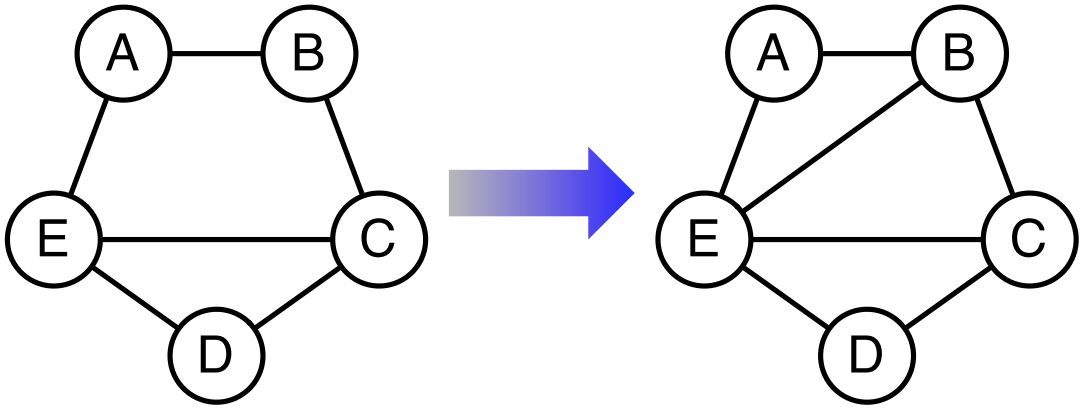
The next step (triangulation) will depend on an undirected graph. Every factor from the DAG must be contained *within* a maximal clique of the undirected graph.

- Replace each factor by an undirected clique (i.e. place edge between every pair of nodes in the factor).
- Construct the potentials on each maximal clique by multiplying together factor potentials that fall within it; ensuring each factor potential only appears once.

The transformation from DAG $\Rightarrow$ undirected graph is called **moralization**:

- "marry" all parents of each node by adding an edge to connect them
- drop arrows on all the edges

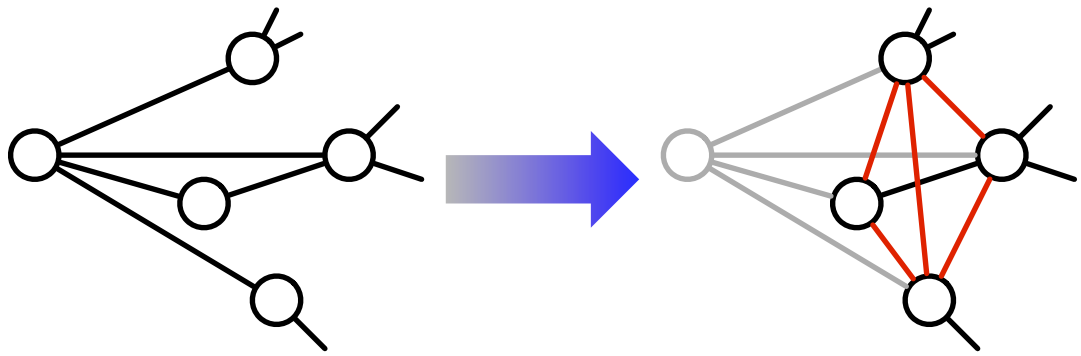
Triangulating the undirected graph



The undirected graph may have loops, which would interfere with belief propagation.

- Could join loops into cliques, but this is inefficient.
- **Triangulation**: add edges to the graph so every loop of size ≥ 4 has at least one chord.
- Recursive: new edges may create new loops; ensure new loops ≥ 4 have chords too.
- An undirected graph where every loop of size ≥ 4 has at least one chord is called **chordal** or **triangulated**.
- Adding edges always *removes* conditional independencies, enlarging the family of distributions that the graph can encode.
- Many ways to add chords; in general finding the best triangulation is NP-complete.
- One approach: **variable elimination**.

Variable elimination

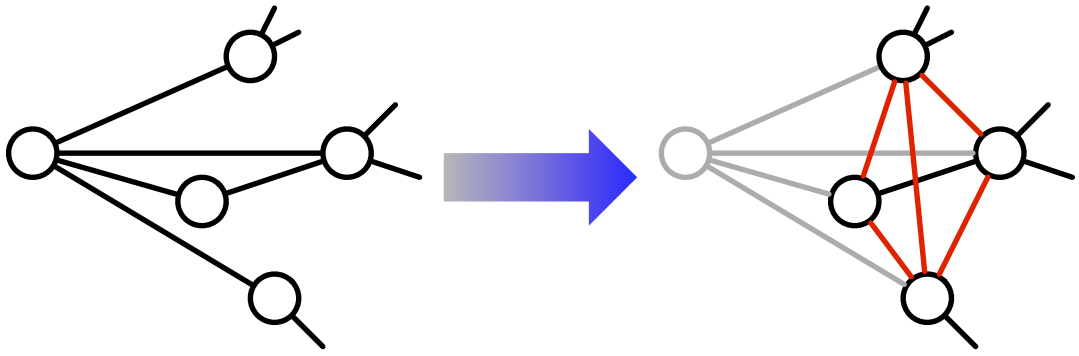


Imagine marginalising the distribution one variable at a time (eliminating each from the graph). Let the order of elimination be $X_{\sigma(1)}, X_{\sigma(2)}, \ldots, X_{\sigma(n)}$:

$$
\begin{aligned}
P(X_{\sigma(n)}) = \sum_{X_{\sigma(n-1)}} \cdots \sum_{X_{\sigma(1)}} P(\mathcal{X}) &= \frac{1}{Z} \sum_{X_{\sigma(n-1)}} \cdots \sum_{X_{\sigma(2)}} \sum_{X_{\sigma(1)}} \prod_i f_i(\mathcal{X}_{C_i}) \\
&= \frac{1}{Z} \sum_{X_{\sigma(n-1)}} \cdots \sum_{X_{\sigma(2)}} \prod_{j : C_i \not\ni \sigma(1)} f_j(\mathcal{X}_{C_j}) \sum_{X_{\sigma(1)}} \prod_{i : C_i \ni \sigma(1)} f_i(\mathcal{X}_{C_i}) \\
&= \frac{1}{Z} \sum_{X_{\sigma(n-1)}} \cdots \sum_{X_{\sigma(2)}} \prod_{j : C_i \not\ni \sigma(1)} f_j(\mathcal{X}_{C_j}) f_{\text{new}}(\mathcal{X}_{C_{\text{new}}})
\end{aligned}
$$

where $C_{\text{new}} = \text{ne}(X_{\sigma(1)})$, and edges are added to the graph connecting all nodes in C_{new}.

Variable elimination



Theorem: a graph including all edges that would be induced by elimination is chordal.

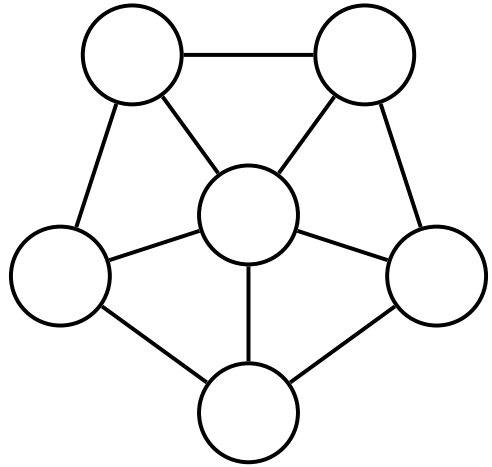
Finding a good triangulation depends on finding a good order of elimination $\sigma(1), \ldots, \sigma(n)$. This is also NP-complete.

Heuristics: pick next variable to eliminate by

- Minimum deficiency search: choose variable that induces the fewest new edges.
- Maximum cardinality search: choose variable with most previously visited neighbours.

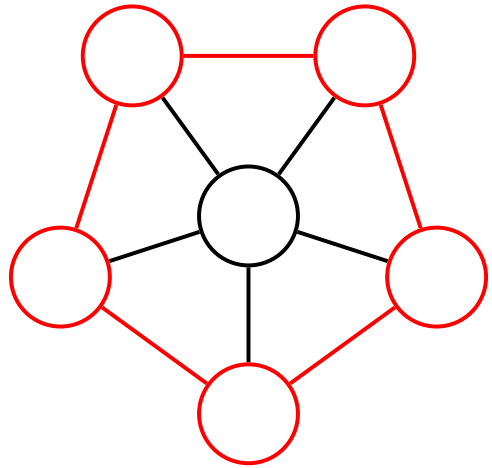
Minimum deficiency search seems (empirically) to be better.

Triangulation may not be obvious by inpection



Is this graph triangulated?

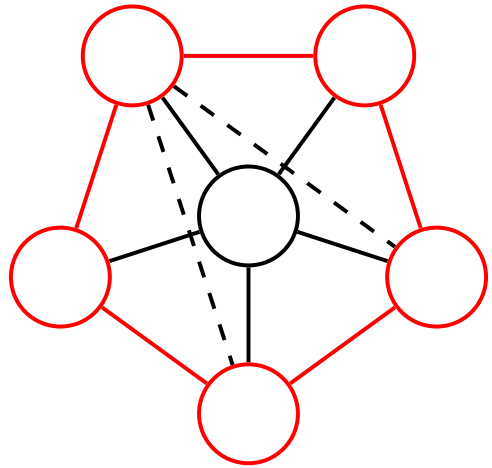
Triangulation may not be obvious by inpection



Is this graph triangulated?

No.

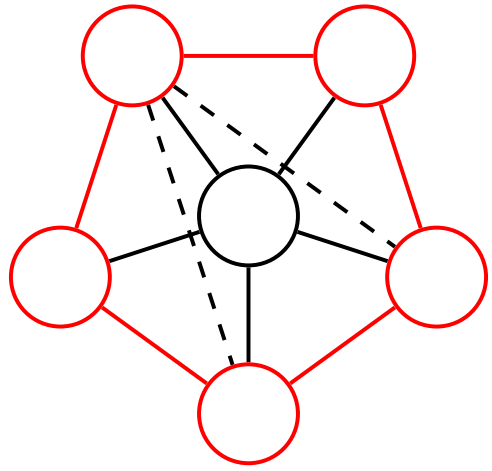
Triangulation may not be obvious by inpection



Is this graph triangulated?

No. Chords must be direct connections — they cannot step through an intermediate node.

Triangulation may not be obvious by inpection

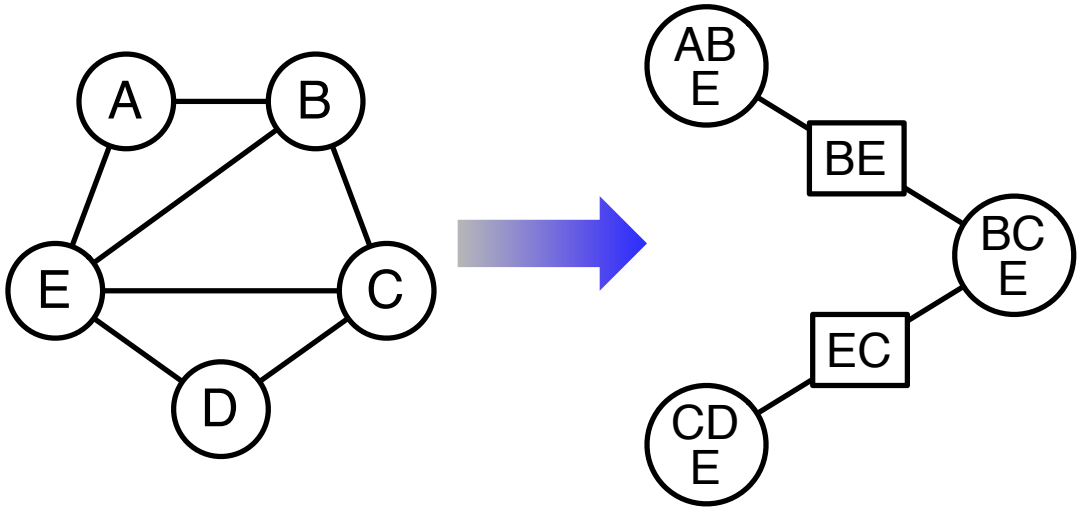


Is this graph triangulated?

No. Chords must be direct connections — they cannot step through an intermediate node.

Detecting unchorded loops by inspection rapidly becomes difficult in large graphs, necessitating automated algorithms such as variable elimination.

Chordal graph to the junction tree



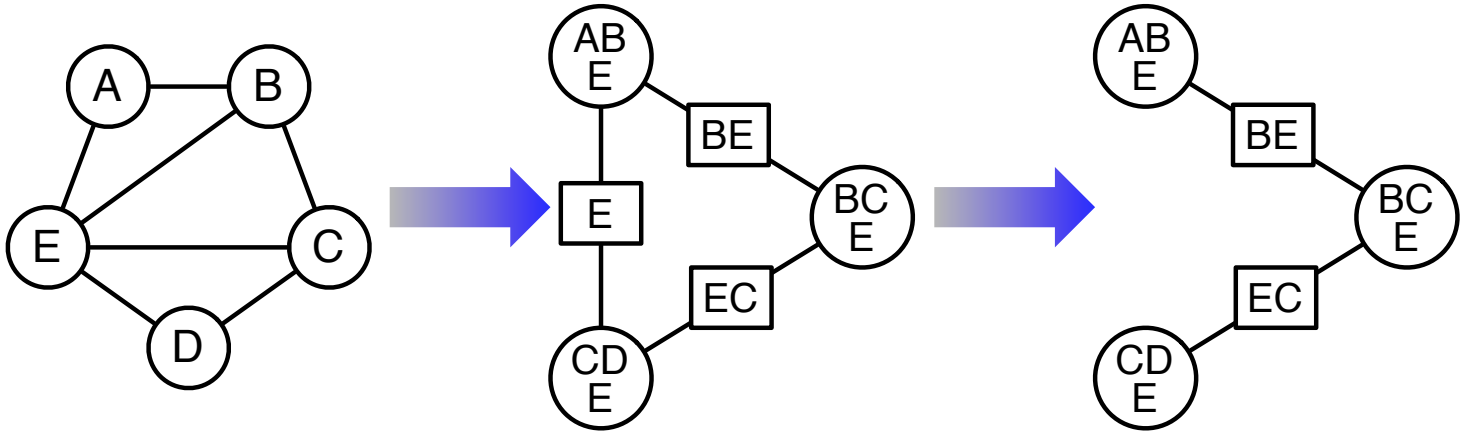
A **junction tree** (or **join tree**) is a tree whose nodes and edges are labelled with sets of variables.

Each node represents a clique; edges are labelled by the intersections of cliques, called separators.

- Cliques contain all adjacent separators.
- **Running intersection property**: if two cliques contain variable X, all cliques and separators on the path between the two cliques contain X.

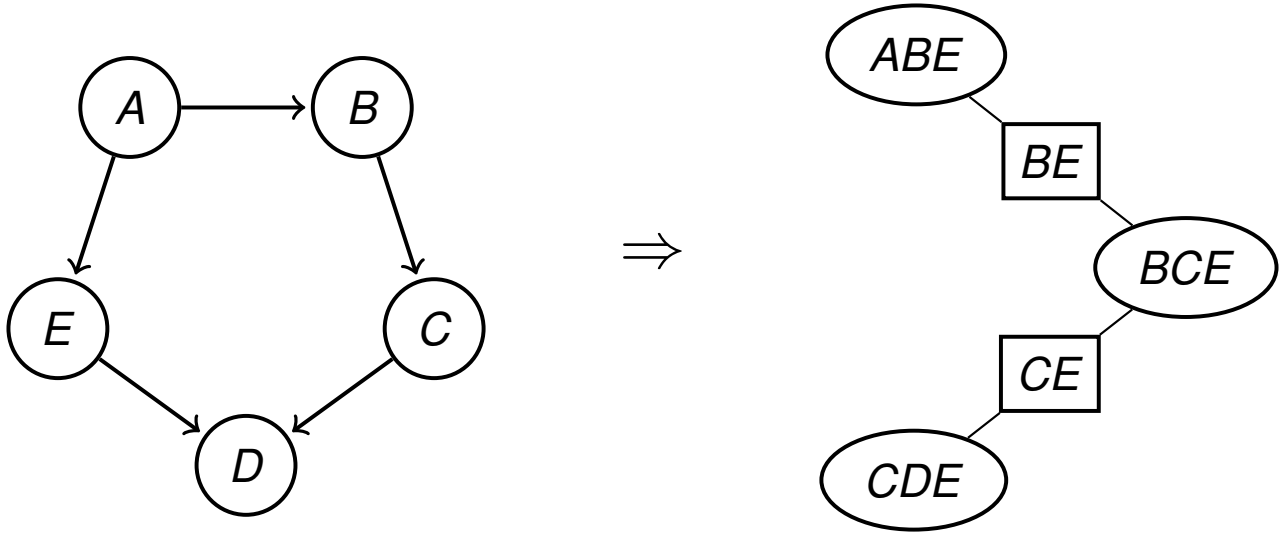
The running intersection property is required for consistency.

Chordal graph to the junction tree



- Find the maximal cliques $C_1, \ldots, C_k$ of the chordal undirected graph (each clique consists of an eliminated variable and its neighbours, so finding maximal cliques is easy).
- Construct a weighted graph, with nodes labelled by the maximal cliques and edges connecting each pair of cliques that shares variables (labelled by the variables in the intersection).
- Define the weight of an edge to be the size of the separator.
- Find the maximum-weight spanning tree of the weighted graph.

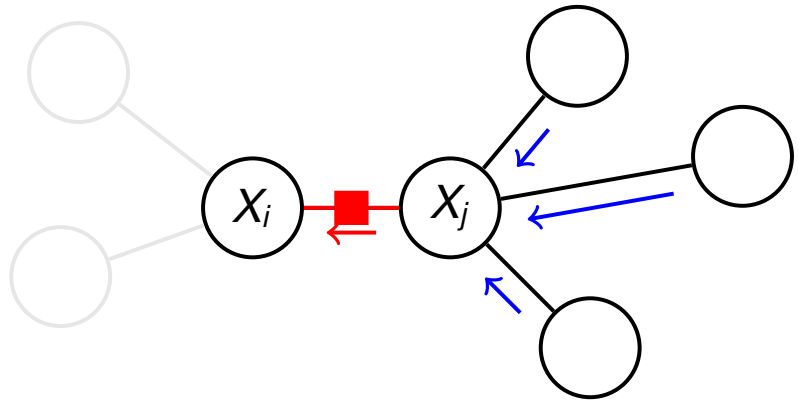
Messages on the junction tree



We've now completed the transformation from a general model to a tree-structured graph.

- Belief propagation on the junction tree should allow us to efficiently compute posterior marginals for inference and learning.

Recall: BP on undirected trees

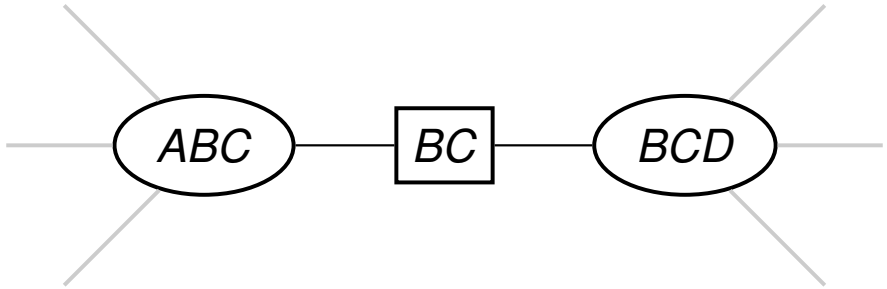


$$M_{j\to i}(X_i) = \sum_{\mathcal{X}_{T_{j\to i}}} f_{(ij)}(X_i, X_j) \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})$$

$$= \sum_{X_j} f_{(ij)}(X_i, X_j) \underbrace{\sum_{\mathcal{X}_{T_{j\to i}}\setminus X_j} \prod_{(i'j')\in\mathcal{E}_{T_{j\to i}}} f_{(i'j')}(X_{i'}, X_{j'})}_{\propto P_{T_{j\to i}}(X_j) \propto \prod_{X_k\in \mathrm{ne}(X_j)\setminus X_i} M_{k\to j}(X_j)}$$

$$= \sum_{X_j} f_{(ij)}(X_i, X_j) \prod_{X_k\in \mathrm{ne}(X_j)\setminus X_i} M_{k\to j}(X_j)$$

Message passing on junction trees

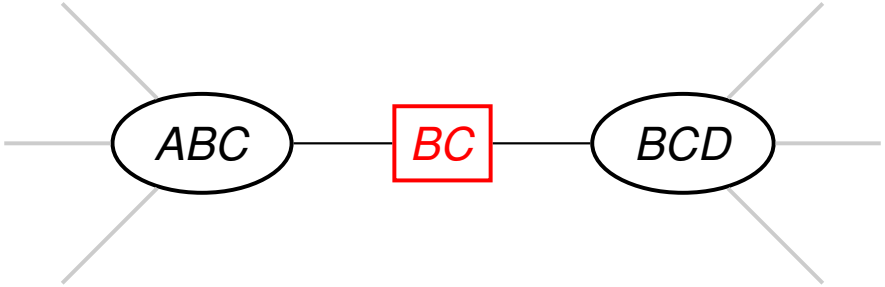


Maximal cliques in the chordal graph are nodes of the junction tree. Thus, the joint distribution factors over the JT nodes:

$$P(\mathcal{X}) = \frac{1}{Z} \prod_i f_i(\mathcal{X}_{C_i}) = \ldots f_{ABC}(A, B, C) f_{BCD}(B, C, D) \ldots$$

This appears to violate the usual undirected tree semantics of a factor per *edge*.

Message passing on junction trees



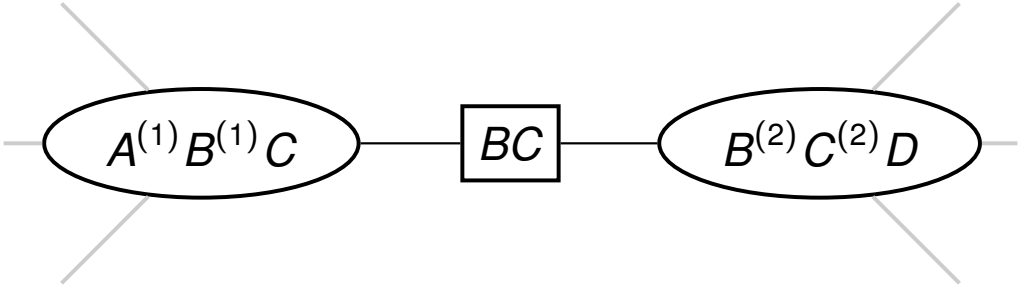
Maximal cliques in the chordal graph are nodes of the junction tree. Thus, the joint distribution factors over the JT nodes:

$$P(\mathcal{X}) = \frac{1}{Z} \prod_i f_i(\mathcal{X}_{C_i}) = \ldots f_{ABC}(A, B, C) f_{BCD}(B, C, D) \ldots$$

This appears to violate the usual undirected tree semantics of a factor per *edge*.

However: the appearance of the same variables in multiple nodes introduces dependencies:

Message passing on junction trees



Maximal cliques in the chordal graph are nodes of the junction tree. Thus, the joint distribution factors over the JT nodes:

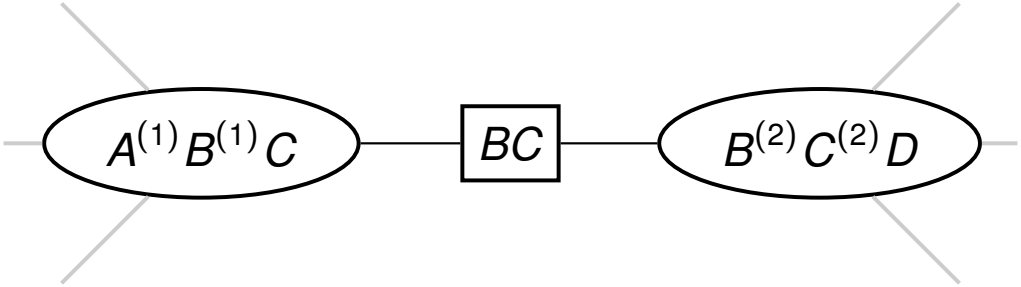
$$P(\mathcal{X}) = \frac{1}{Z}\prod_i f_i(\mathcal{X}_{C_i}) = \ldots f_{ABC}(A,B,C)f_{BCD}(B,C,D)\ldots$$

This appears to violate the usual undirected tree semantics of a factor per *edge*.

However: the appearance of the same variables in multiple nodes introduces dependencies:

- Introduce copy variables on each side of the separator.

Message passing on junction trees



Maximal cliques in the chordal graph are nodes of the junction tree. Thus, the joint distribution factors over the JT nodes:

$$P(\mathcal{X}) = \frac{1}{Z}\prod_i f_i(\mathcal{X}_{C_i}) = \ldots f_{ABC}(A,B,C)f_{BCD}(B,C,D)\ldots$$

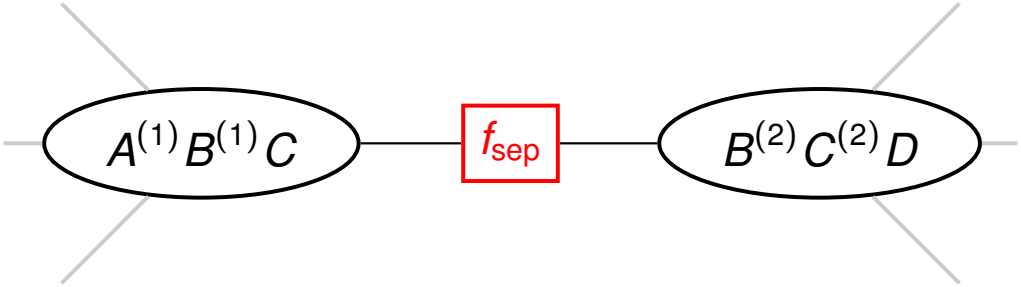
This appears to violate the usual undirected tree semantics of a factor per *edge*.

However: the appearance of the same variables in multiple nodes introduces dependencies:

- Introduce copy variables on each side of the separator.
- Factors on nodes no longer overlap.

$$P(\mathcal{X}) = \ldots f_{ABC}(A,B^{(1)},C^{(1)}) \qquad f_{BCD}(B^{(2)},C^{(2)},D)\ldots$$

Message passing on junction trees



Maximal cliques in the chordal graph are nodes of the junction tree. Thus, the joint distribution factors over the JT nodes:

$$P(\mathcal{X}) = \frac{1}{Z}\prod_i f_i(\mathcal{X}_{C_i}) = \ldots f_{ABC}(A,B,C) f_{BCD}(B,C,D)\ldots$$

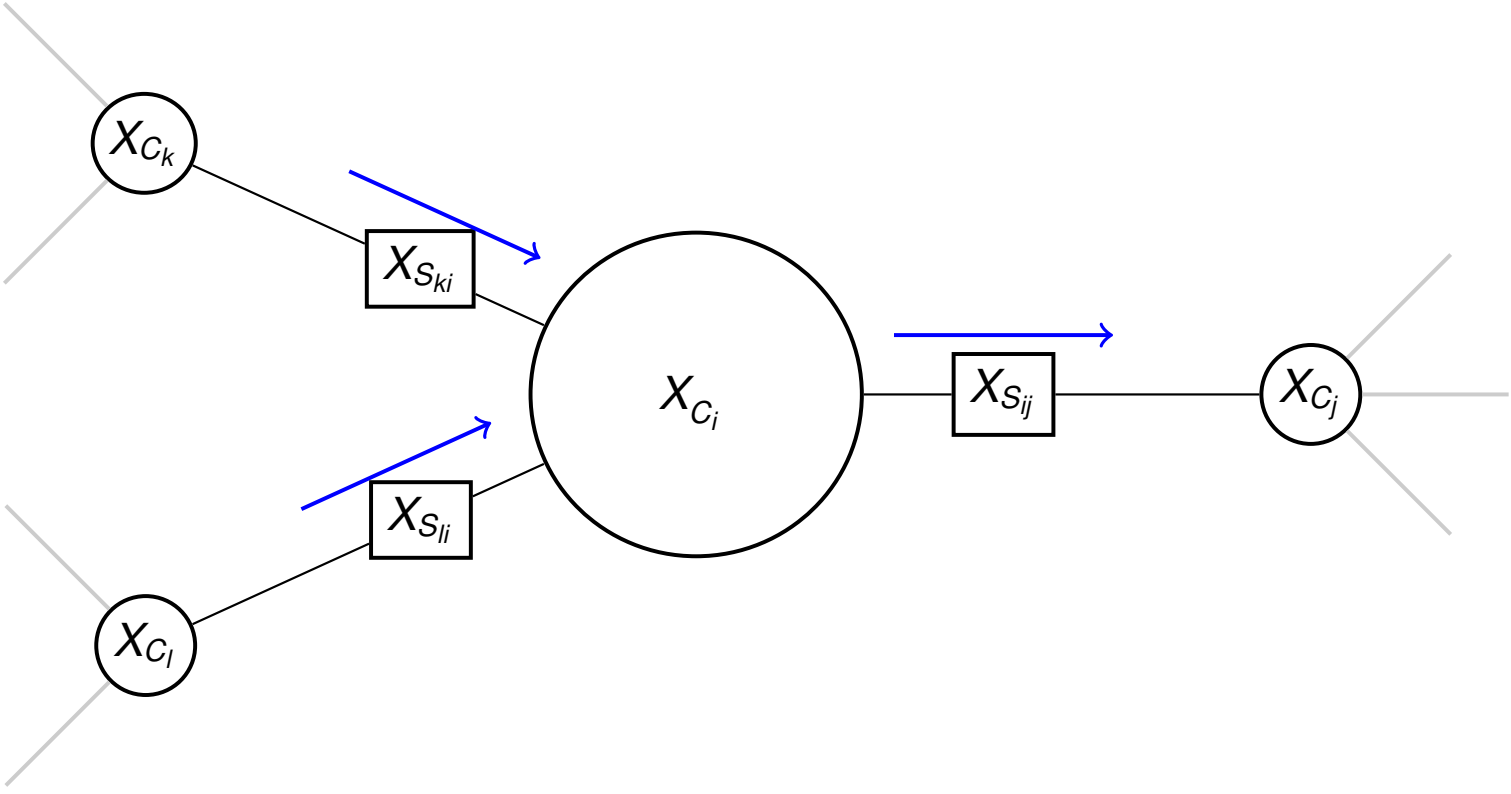
This appears to violate the usual undirected tree semantics of a factor per *edge*.

However: the appearance of the same variables in multiple nodes introduces dependencies:

- Introduce copy variables on each side of the separator.
- Factors on nodes no longer overlap.
- New delta-function factors on separators enforce consistency amongst copies:

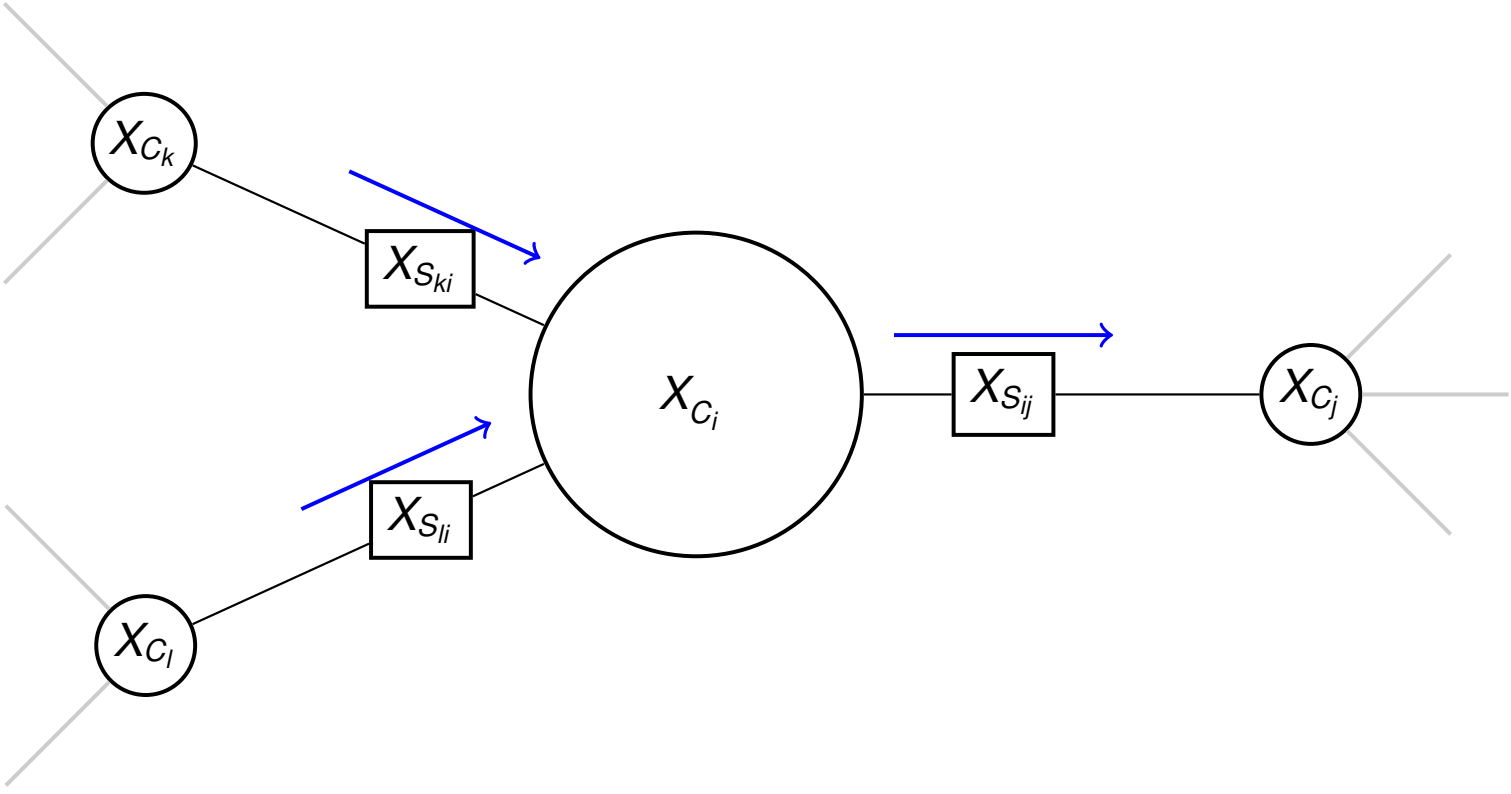
$$P(\mathcal{X}) = \ldots f_{ABC}(A,B^{(1)},C^{(1)}) \underbrace{\delta(B^{(1)}-B^{(2)})\delta(C^{(1)}-C^{(2)})}_{f_{\text{sep}}(B^{(1)},C^{(1)},B^{(2)},C^{(2)})} f_{BCD}(B^{(2)},C^{(2)},D)\ldots$$

Message passing on junction trees



We can use this view to define BP messages on the junction tree:

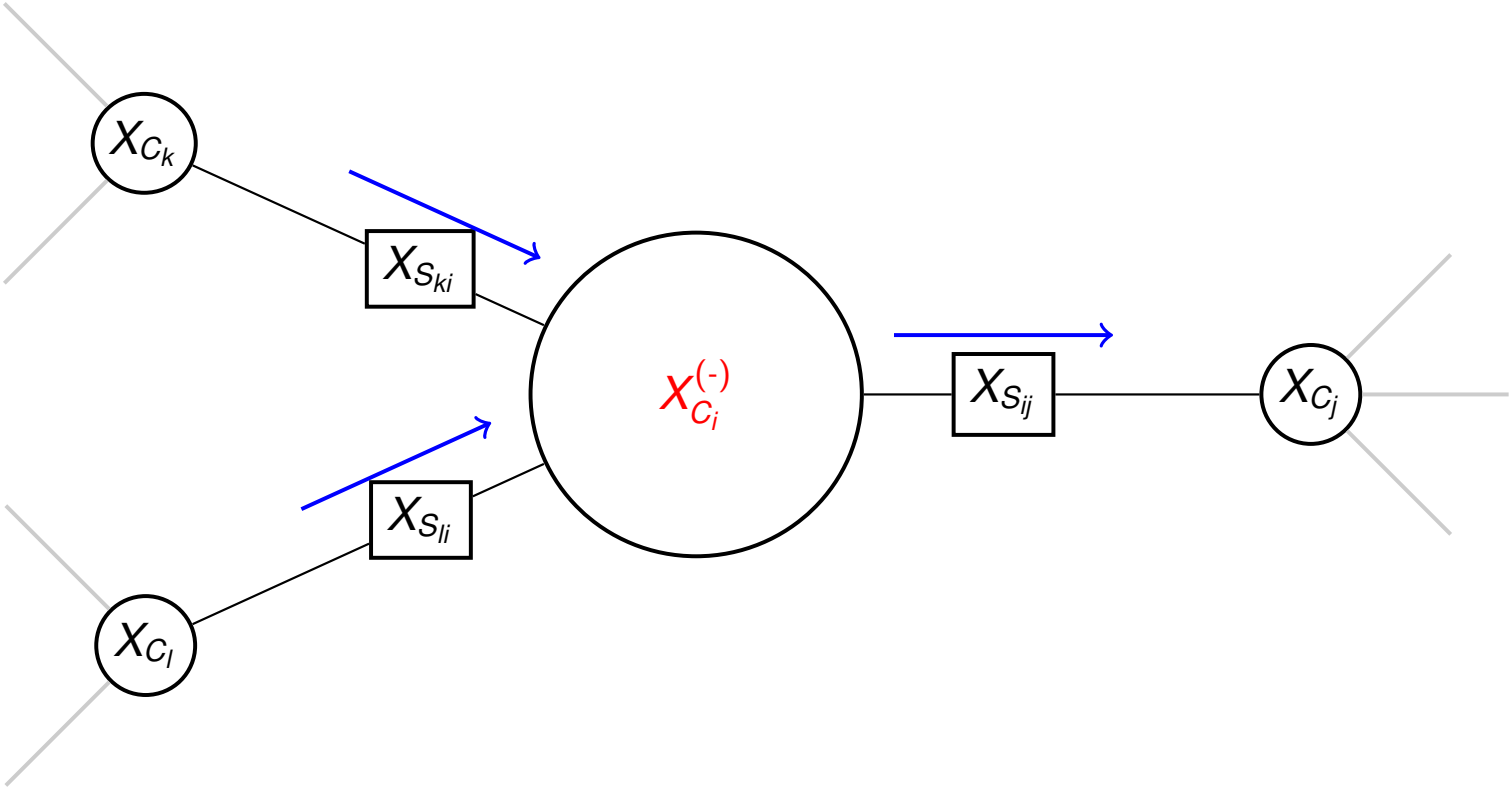
Message passing on junction trees



We can use this view to define BP messages on the junction tree:

- Copy and partition clique variables X_{C_i}:

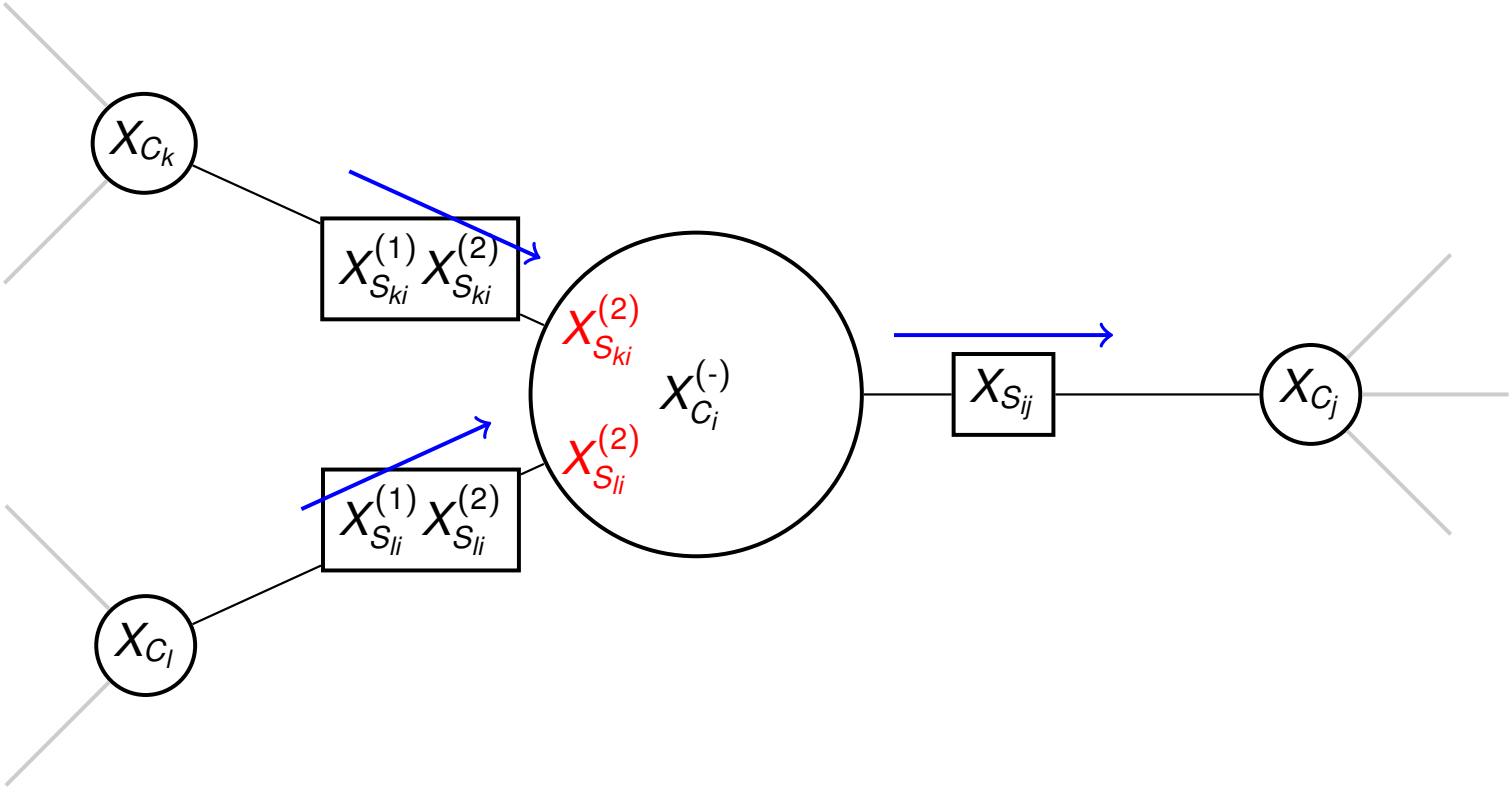
Message passing on junction trees



We can use this view to define BP messages on the junction tree:

- Copy and partition clique variables X_{C_i}:
  - Unshared variables: $X_{C_i}^{(-)} = X_{C_i} \setminus \bigcup S_{ik}$

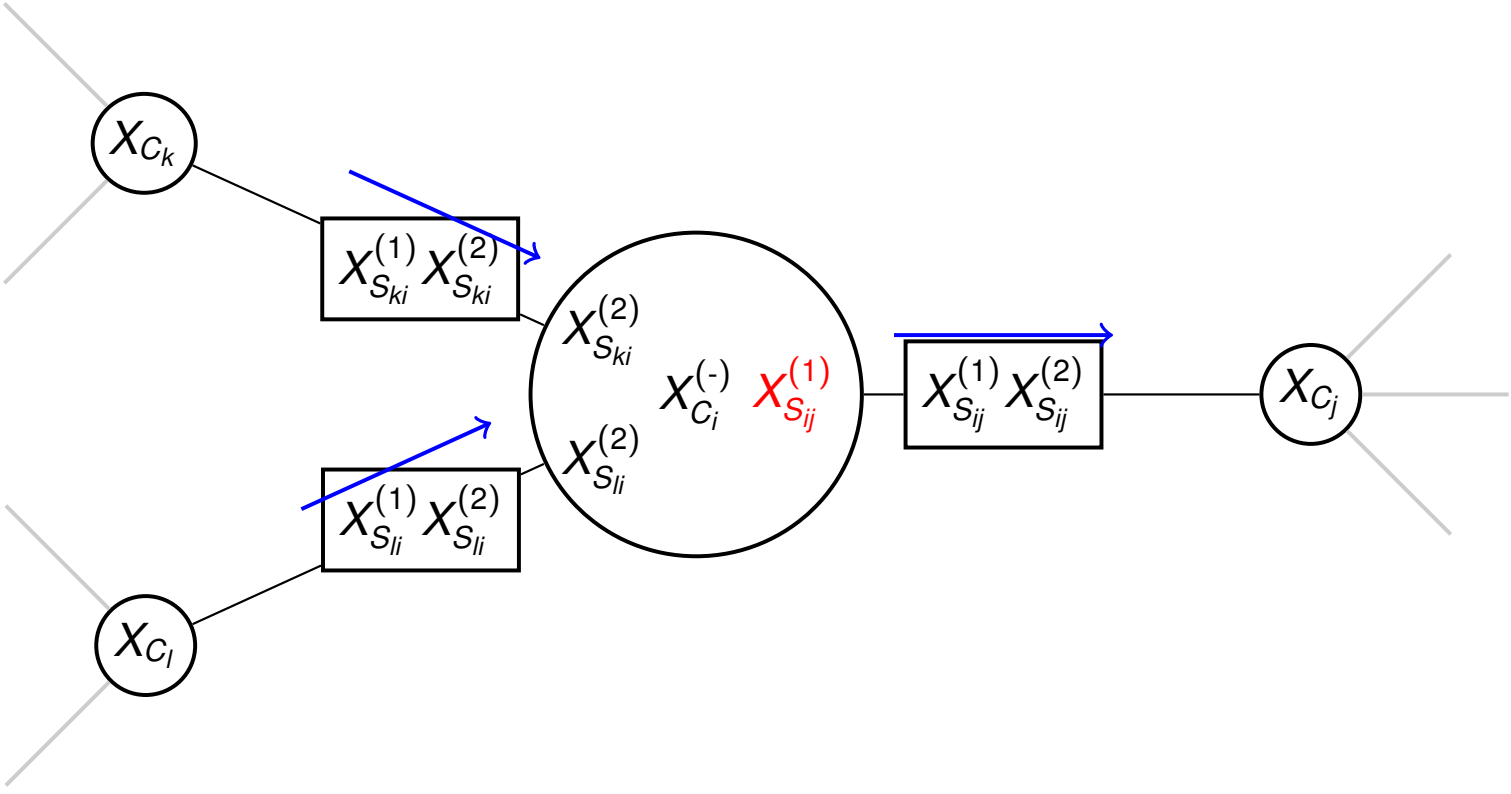
Message passing on junction trees



We can use this view to define BP messages on the junction tree:

- Copy and partition clique variables X_{C_i}:
  - Unshared variables: $X^{(-)}_{C_i} = X_{C_i \setminus \bigcup S_{ik}}$
  - Variables in incoming separators: $X^{(2)}_{S_{ki}}$ (matching variables $X^{(1)}_{S_{ki}}$ in $k \in \mathrm{ne}(i) \setminus j$).

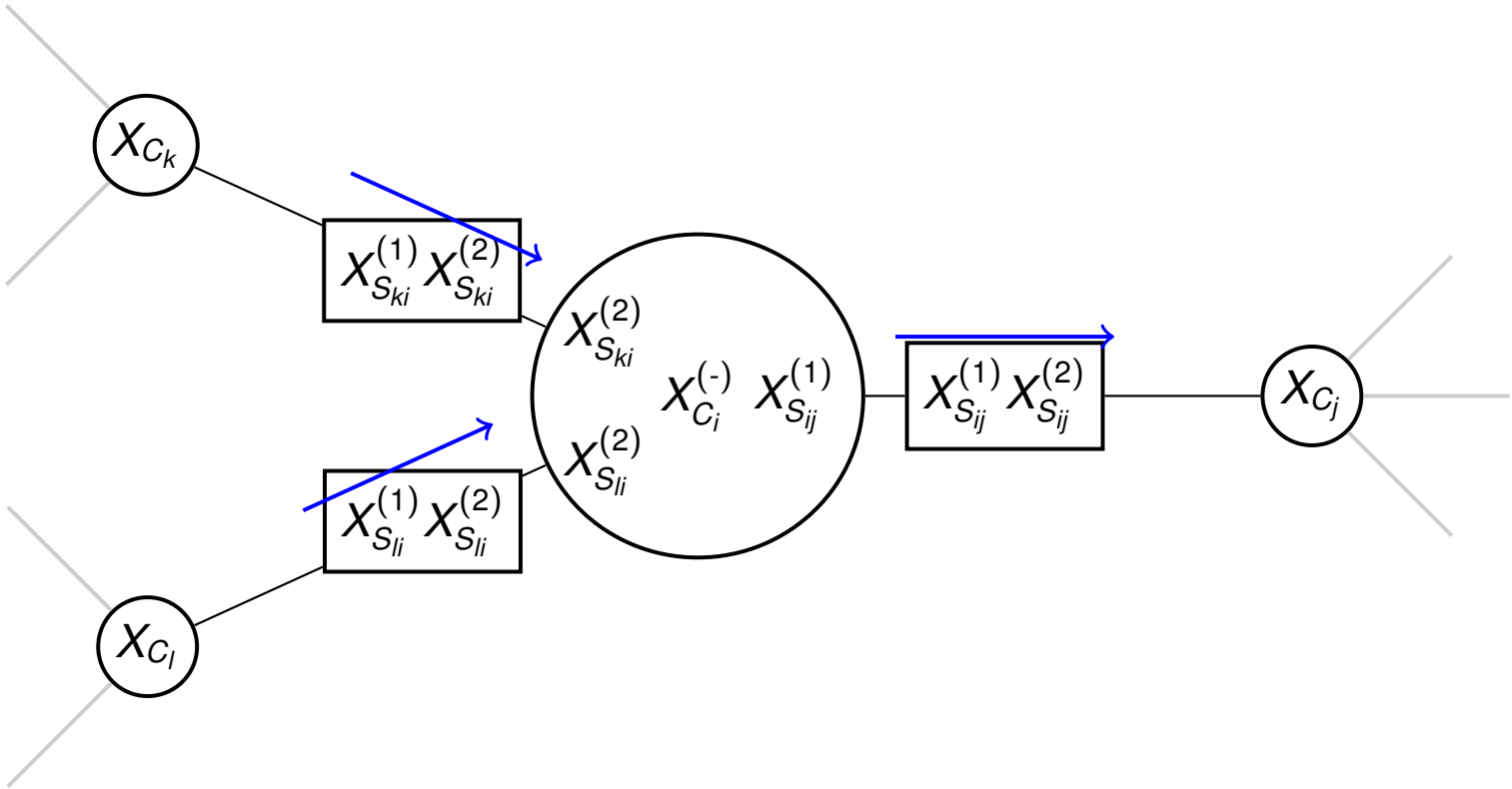
Message passing on junction trees



We can use this view to define BP messages on the junction tree:

- Copy and partition clique variables X_{C_i}:
  - Unshared variables: $X^{(-)}_{C_i} = X_{C_i \setminus \bigcup S_{ik}}$
  - Variables in incoming separators: $X^{(2)}_{S_{ki}}$ (matching variables $X^{(1)}_{S_{ki}}$ in $k \in \mathrm{ne}(i) \setminus j$).
  - Variables in outgoing separator: $X^{(1)}_{S_{ij}}$ (matching variables $X^{(2)}_{S_{ij}}$ in clique j).

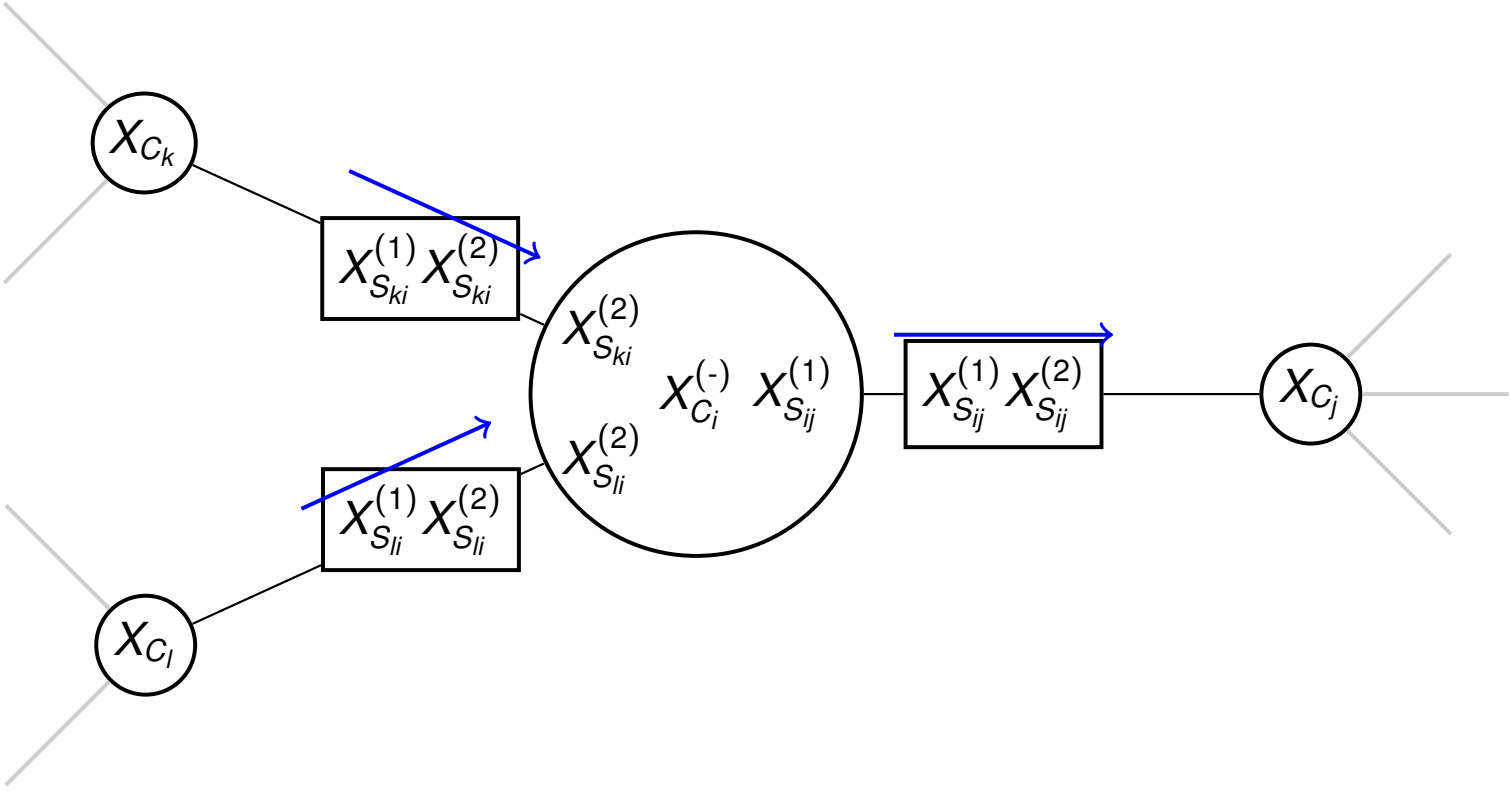
Message passing on junction trees



We can use this view to define BP messages on the junction tree:

- Copy and partition clique variables X_{C_i}:
  - Unshared variables: $X^{(-)}_{C_i} = X_{C_i \setminus \bigcup s_{ik}}$
  - Variables in incoming separators: $X^{(2)}_{S_{ki}}$ (matching variables $X^{(1)}_{S_{ki}}$ in $k \in \mathrm{ne}(i) \setminus j$).
  - Variables in outgoing separator: $X^{(1)}_{S_{ij}}$ (matching variables $X^{(2)}_{S_{ij}}$ in clique j).
  - (Variables that appear in more than one separator will need additional copies.)

Message passing on junction trees

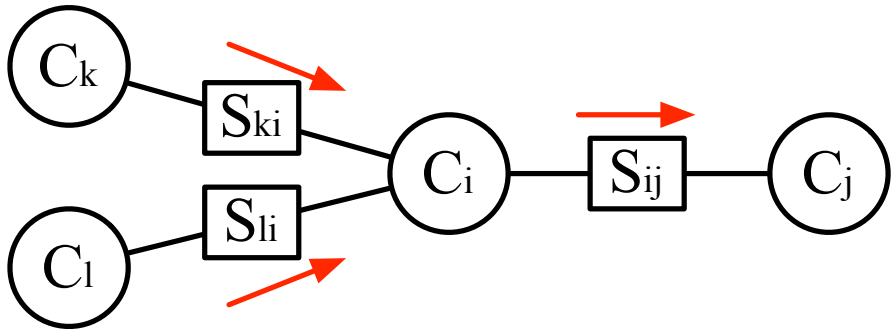


Messages become:

$$M_{i\to j}(X_{S_{ij}}^{(2)}) = \sum_{X_{C_i}^{(-)},\{X_{S_{ki}}^{(2)}\},X_{S_{ij}}^{(1)}} f_i(X_{C_i}^{(-)}, \{X_{S_{ki}}^{(2)}\}, X_{S_{ij}}^{(1)}) f_{ij}(X_{S_{ij}}^{(1)}, X_{S_{ij}}^{(2)}) \prod_{k\in \mathrm{ne}(i)\setminus j} M_{k\to i}(X_{S_{ki}}^{(2)})$$

$$= \sum_{X_{C_i\setminus S_{ij}}} f_i(X_{C_i}) \prod_{k\in \mathrm{ne}(i)\setminus j} M_{k\to i}(X_{S_{ki}})$$

Shafer-Shenoy propagation



Messages computed recursively by:

$$M_{i\to j}(X_{S_{ij}}) = \sum_{X_{C_i}\setminus S_{ij}} f_i(X_{C_i}) \prod_{k\in \mathrm{ne}(i)\setminus j} M_{k\to i}(X_{S_{ki}})$$

And marginal distributions on cliques and separators are:

$$P(X_{C_i}) = f_i(X_{C_i}) \prod_{k\in \mathrm{ne}(i)} M_{k\to i}(X_{S_{ki}})$$

$$P(X_{S_{ij}}) = M_{i\to j}(X_{S_{ij}}) M_{j\to i}(X_{S_{ij}})$$

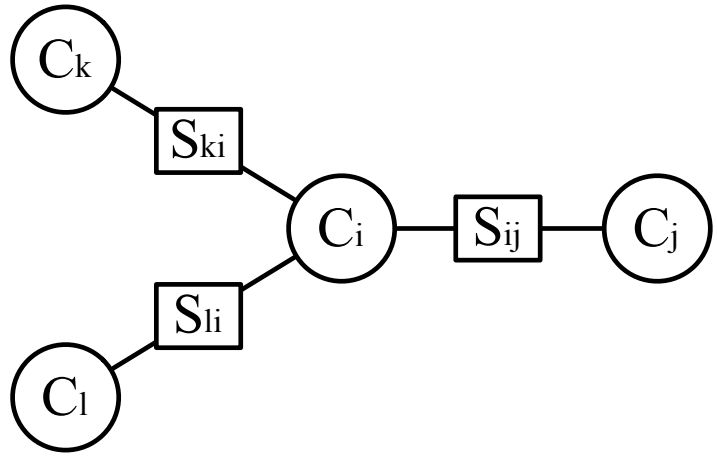
This is called **Shafer-Shenoy propagation**.

Consistency

The running intersection property and tree structure of the junction trees implies that **local consistency** between cliques and separator marginals guarantees **global consistency**.

If $q_i(X_{C_i})$, $r_{ij}(X_{S_{ij}})$ are distributions such that

$$\sum_{X_{C_i} \setminus S_{ij}} q_i(X_{C_i}) = r_{ij}(X_{S_{ij}})$$



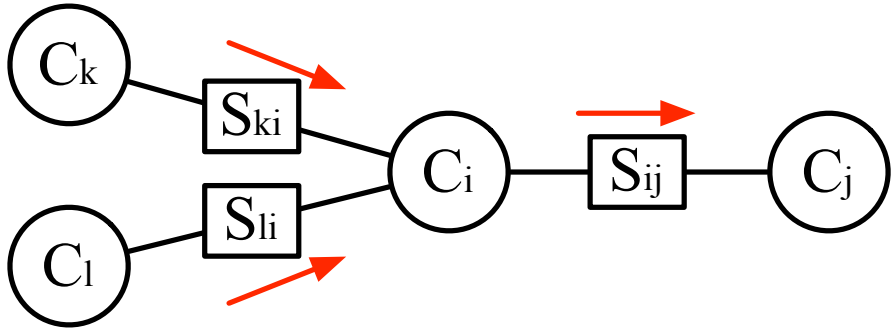
Then the following

$$P(\mathcal{X}) = \frac{\prod_{\text{cliques } i} q_i(X_{C_i})}{\prod_{\text{separators } (ij)} r_{ij}(X_{S_{ij}})}$$

is also a distribution (non-negative and sums to one) such that:

$$q_i(X_{C_i}) = \sum_{\mathcal{X} \setminus X_{C_i}} P(\mathcal{X}) \qquad r_{ij}(X_{S_{ij}}) = \sum_{\mathcal{X} \setminus X_{S_{ij}}} P(\mathcal{X})$$

Reparameterisation for message passing



Hugin propagation is a different (but equivalent) message passing algorithm.
It is based upon the idea of **reparameterisation**. Initialize:

$$q_i(X_{C_i}) \propto f_i(X_{C_i}) \qquad r_{ij}(X_{S_{ij}}) \propto 1$$

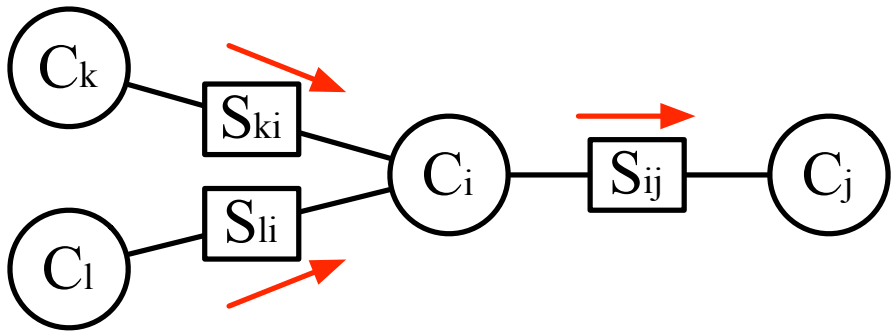
Then our probability distribution is initially

$$P(\mathcal{X}) \propto \frac{\prod_{\text{cliques } i} q_i(X_{C_i})}{\prod_{\text{separators } (ij)} r_{ij}(X_{S_{ij}})}$$

A Hugin propagation update for $i \rightarrow j$ is:

$$r_{ij}^{\text{new}}(X_{S_{ij}}) = \sum_{X_{C_i} \setminus S_{ij}} q_i(X_{C_i}) \qquad q_j^{\text{new}}(X_{C_j}) = q_j(X_{C_j}) \frac{r_{ij}^{\text{new}}(X_{S_{ij}})}{r_{ij}(X_{S_{ij}})}$$

Hugin propagation



Some properties of Hugin propagation:

- The defined distribution $P(\mathcal{X})$ is unchanged by the updates.
- Each update introduces a local consistency constaint:

$$\sum_{X_{C_j \setminus S_{ij}}} q_j(X_{C_j}) = r_{ij}(X_{S_{ij}})$$

- If each update $i \to j$ is carried out only after incoming updates $k \to i$ have been carried out, then each update needs only be carried out once.
- Each Hugin update is equivalent to the corresponding Shafer-Shenoy update.

Computational Costs of the Junction Tree Algorithm

- Most of the computational cost of the junction tree algorithm is incurred during the message passing phase.
- The running and memory costs of the message passing phase are both $O(\sum_i |\mathcal{X}_{C_i}|)$. This can be significantly (exponentially) more efficient than brute force.
- The variable elimination ordering heuristic can have very significant impact on the message passing costs.
- For certain classes of graphical models (e.g. 2D lattice Markov random field) it is possible to hand-craft an efficient ordering.

Other Inference Algorithms

There are other approaches to inference in graphical models which may be more efficient under specific conditions:

Cutset conditioning: or "reasoning by assumptions". Find a small set of variables which, if they were given (i.e. known) would render the remaining graph "simpler". For each value of these variables run some inference algorithm on the simpler graph, and average the resulting beliefs with the appropriate weights.

Loopy belief propagation: just use belief propagation even though there are loops. No guarantee of convergence, but often works well in practice. Some (weak) guarantees about the nature of the answer if the message passing *does* converge.

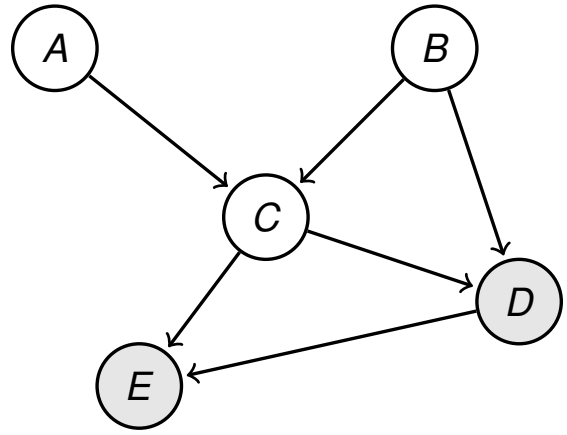
Second half of course: we will learn about a variety of approximate inference algorithms when the graphical model is so large/complex that no exact inference algorithm can work efficiently.

Learning in Graphical Models

We have discussed inference at length — what about learning? The factored structure implied by the graph also makes learning easy.

Consider data points comprising observations of a subset of variables.
ML learning $\Rightarrow$ adjust parameters to maximise:

$$\begin{aligned}\mathcal{L} &= P(X_{\text{obs}}|\theta) \\ &= \sum_{X_{\text{unobs}}} P(X_{\text{obs}}, X_{\text{unobs}}|\theta)\end{aligned}$$



by EM, need to maximise

$$\begin{aligned}\mathcal{F}(q,\theta) &= \langle \log P(X_{\text{obs}}, X_{\text{unobs}}|\theta) - \log q(X_{\text{unobs}}) \rangle_{q(X_{\text{unobs}})} \\ &= \left\langle \textstyle\sum_i \log f_i(X_{C_i}|\theta_i) - \log Z(\theta) \right\rangle_{q(X_{\text{unobs}})} + \mathbf{H}(q) \\ &= \sum_i \langle \log f_i(X_{C_i}|\theta_i) \rangle_{q(X_{C_i} \setminus X_{\text{obs}})} - \log Z(\theta) + \mathbf{H}(q)\end{aligned}$$

So learning only requires posterior marginals on cliques (obtained by messaging passing) and updates on cliques; c.f. the Baum-Welch procedure for HMMs.