Unsupervised Learning

Graphical Models

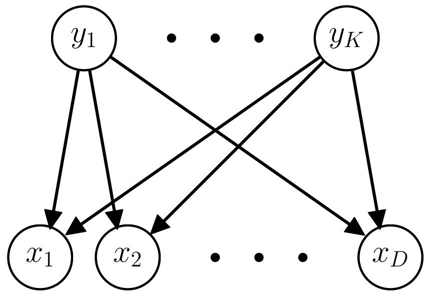
Maneesh Sahani

maneesh@gatsby.ucl.ac.uk

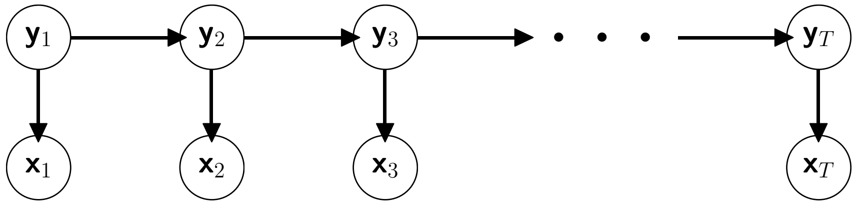
**Gatsby Computational Neuroscience Unit, and
MSc in Intelligent Systems, Dept Computer Science
University College London**

Term 1, Autumn 2006

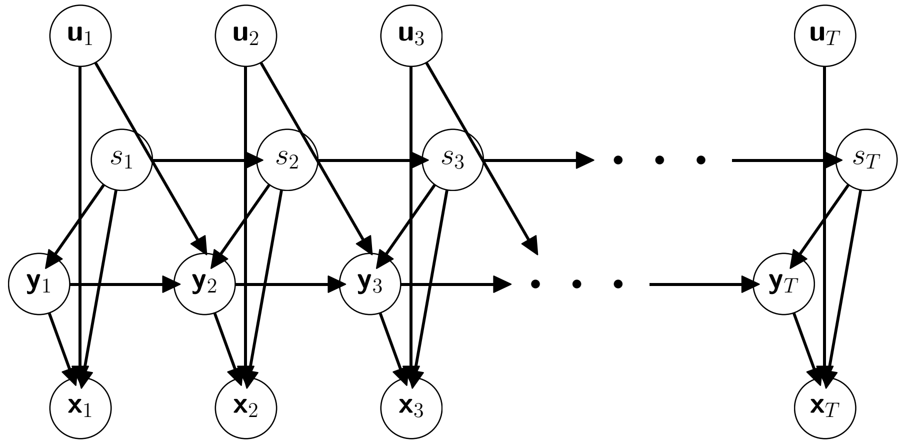
Some Examples



factor analysis
probabilistic PCA

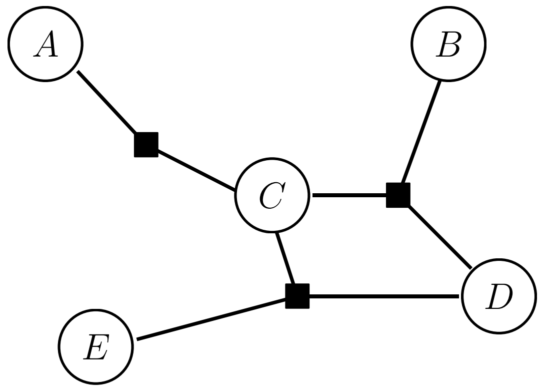


hidden Markov models
linear dynamical systems

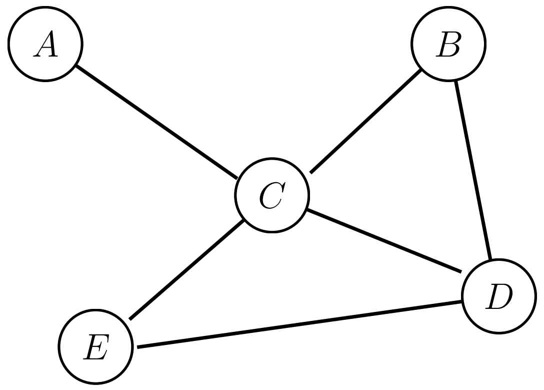


switching state-space models

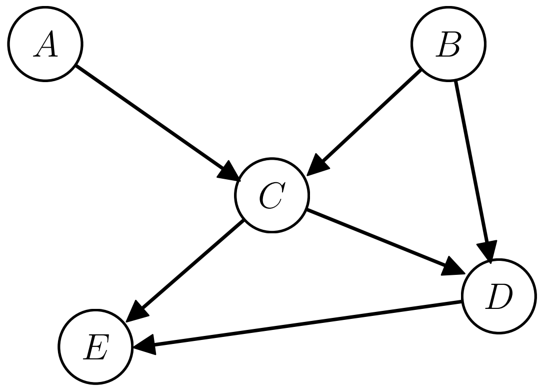
Three kinds of graphical models



factor graph



undirected graph



directed graph

Why do we need graphical models?

- Graphs are an **intuitive** way of representing and visualising the relationships between many variables. (Examples: family trees, electric circuit diagrams, neural networks)
- A graph allows us to abstract out the **conditional independence** relationships between the variables from the details of their parametric forms. Thus we can ask questions like: "Is A dependent on B given that we know the value of C ?" just by looking at the graph.
- Graphical models allow us to define general **message-passing algorithms** that implement Bayesian inference efficiently. Thus we can answer queries like "What is $P(A|C = c)$?" without enumerating all settings of all variables in the model.

Conditional Independence

Conditional Independence:

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

when $p(Y,V) > 0$. Also

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

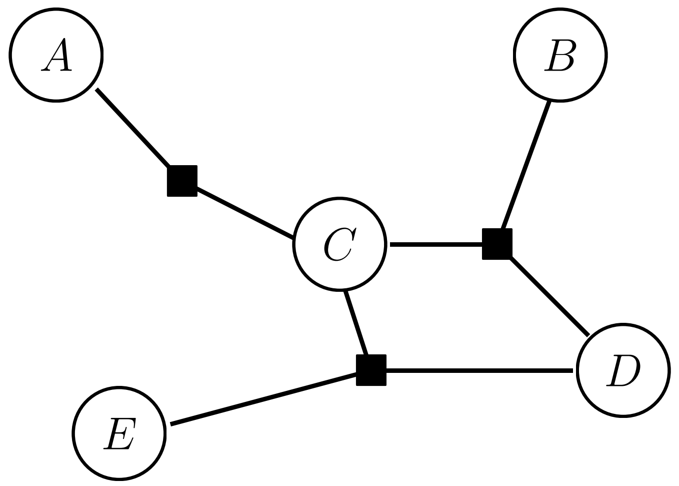
In general we can think of conditional independence between **sets of variables**:

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

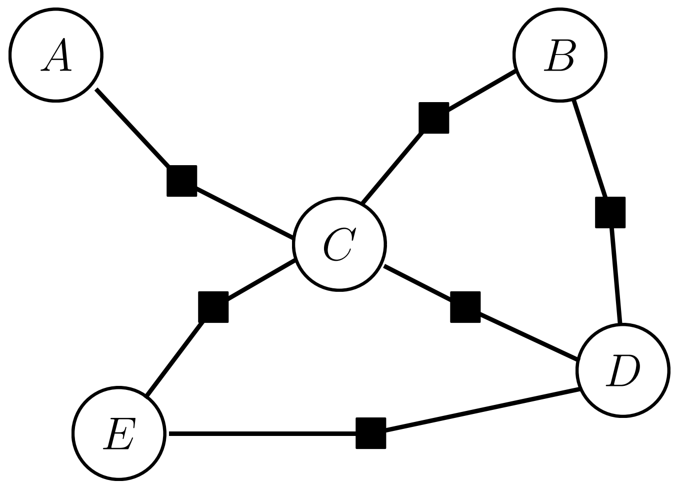
Marginal Independence:

$$X \perp\!\!\!\perp Y \quad \Leftrightarrow \quad X \perp\!\!\!\perp Y | \emptyset \quad \Leftrightarrow \quad p(X,Y) = p(X)p(Y)$$

Factor Graphs



(a)



(b)

The circles in a factor graph represent random variables.
The filled dots represent factors in the joint distribution.

(a) $P(A,B,C,D,E) = \frac{1}{Z} g_1(A,C) g_2(B,C,D) g_3(C,D,E)$

(b) $P(A,B,C,D,E) = \frac{1}{Z} g_1(A,C) g_2(B,C) g_3(C,D) g_4(B,D) g_5(C,E) g_6(D,E)$

The g_i are non-negative functions of their arguments, and Z is a normalization constant.
Two nodes are neighbours if they share a common factor.

Fact: $X \perp\!\!\!\perp Y | \mathcal{V}$ if every path between X and Y contains some node $V \in \mathcal{V}$
Corollary: Given the neighbours of X, the variable X is conditionally independent of all other variables: $X \perp\!\!\!\perp Y | \, \mathrm{ne}(X), \quad \forall Y \notin \{X \cup \mathrm{ne}(X)\}$

Undirected Graphical Models

In an Undirected Graphical Model, the joint probability over all variables can be written in a factored form:

$$P(\mathbf{x}) = \frac{1}{Z} \prod_j g_j(\mathbf{x}_{C_j})$$

where $\mathbf{x} = (x_1, \ldots, x_K)$, and

$$C_j \subseteq \{1, \ldots, K\}$$

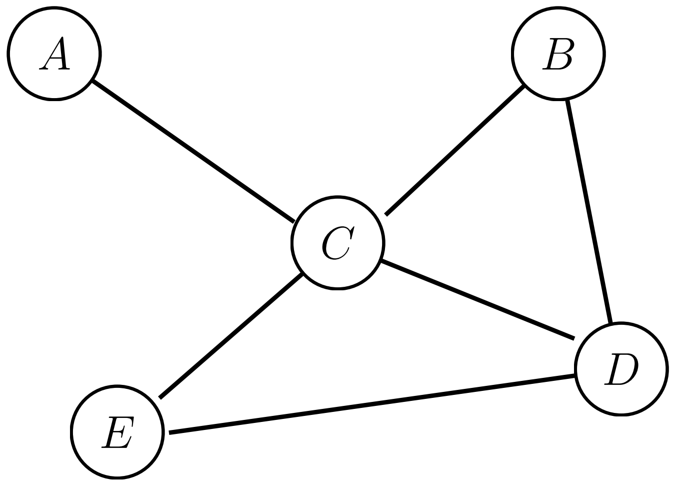
are subsets of the set of all variables, and $\mathbf{x}_S \equiv (x_k : k \in S)$.

This type of probabilistic model can be represented graphically.

Graph Definition: Let each variable be a node. Connect nodes i and k if there exists a set C_j such that both $i \in C_j$ and $k \in C_j$. These sets form the *cliques* of the graph (fully connected subgraphs).

Note: Undirected Graphical Models are also called *Markov Networks*.

Undirected Graphical Models



$$P(A,B,C,D,E) = \frac{1}{Z} g_1(A,C) g_2(B,C,D) g_3(C,D,E)$$

Fact: $X \perp\!\!\!\perp Y | \mathcal{V}$ if every path between X and Y contains some node $V \in \mathcal{V}$

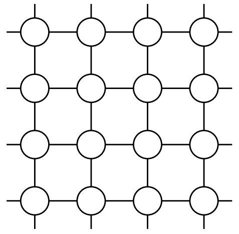
Corollary: Given the neighbours of X, the variable X is conditionally independent of all other variables: $X \perp\!\!\!\perp Y \,|\, \mathrm{ne}(X), \quad \forall Y \notin \{X \cup \mathrm{ne}(X)\}$

Markov Blanket: $\mathcal{V}$ 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 $\equiv$ $\mathrm{ne}(X)$ for undirected graphs and factor graphs

Examples of Undirected Graphical Models

- Markov Random Fields (used in Computer Vision)



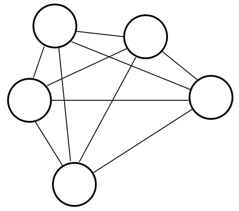
- Exponential Language Models (used in Speech and Language Modelling)

$$p(s) = \frac{1}{Z} p_0(s) \exp \left\{ \sum_i \lambda_i f_i(s) \right\}$$

- Products of Experts (widely applicable)

$$p(\mathbf{x}) = \frac{1}{Z} \prod_j p_j(\mathbf{x}|\theta_j)$$

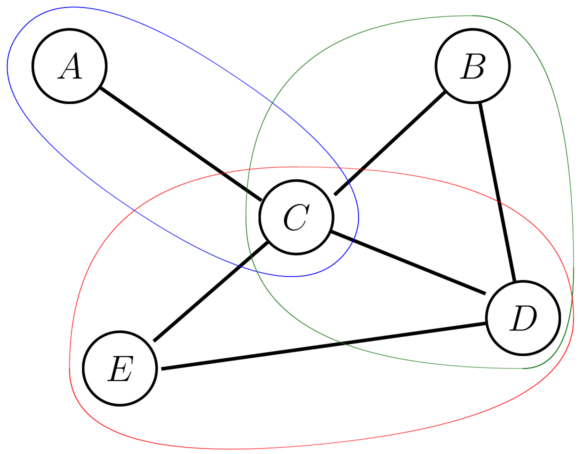
- Boltzmann Machines (a kind of Neural Network/Ising Model)



Clique Potentials and Undirected Graphs

Definition: a *clique* is a fully connected subgraph. By clique we usually mean maximal clique (i.e. not contained within another clique)

C_i denotes the set of variables in the i^{th} clique.



$$p(x_1, \dots, x_K) = \frac{1}{Z} \prod_i g_i(\mathbf{x}_{C_i})$$

where $Z = \sum_{x_1 \cdots x_K} \prod_i g_i(\mathbf{x}_{C_i})$ is the normalization.

Associated with each clique C_i is a non-negative function $g_i(\mathbf{x}_{C_i})$ which measures "compatibility" between settings of the variables.

A	C	$g_1(A, C)$
0	0	0.2
0	1	0.6
1	0	0.0
1	1	1.2

Example: Let $C_1 = \{A, C\}, A \in \{0, 1\}, C \in \{0, 1\}$
What does this mean?

Hammersley–Clifford Theorem (1971)

Theorem: A probability function p formed by a normalized product of positive functions on cliques of G is a Markov Field relative to G.

Definition: The distribution p is a *Markov Field relative to* G if all conditional independence relations represented by G are true of p.

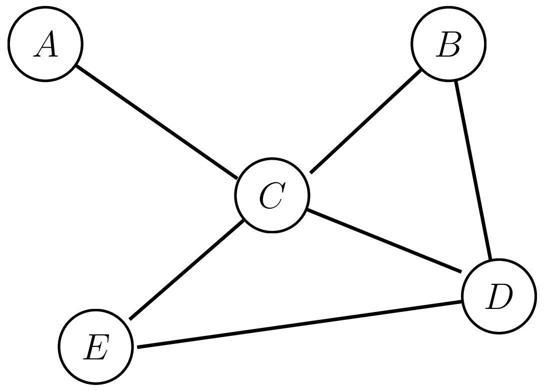
G represents the following CI relations: If $V \in \mathcal{V}$ lies on *all* paths between X and Y in G, then $X \perp\!\!\!\perp Y | \mathcal{V}$.

Proof: We need to show that if p is a product of functions on cliques of G then a variable is conditionally independent of its non-neighbours in G given its neighbours in G. That is: $\mathrm{ne}(x_\ell)$ is a Markov Blanket for x_ℓ. Let $x_m \notin \{x_\ell \cup \mathrm{ne}(x_\ell)\}$

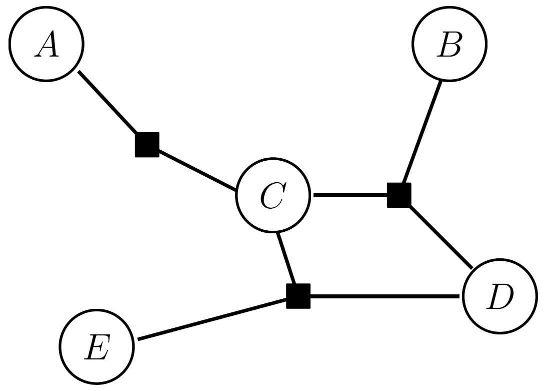
$$\begin{aligned} p(x_\ell, x_m, \ldots) &= \frac{1}{Z} \prod_i g_i(\mathbf{x}_{C_i}) = \frac{1}{Z} \prod_{i:\ell \in C_i} g_i(\mathbf{x}_{C_i}) \prod_{j:\ell \notin C_j} g_j(\mathbf{x}_{C_j}) \\ &= \frac{1}{Z'} f_1\big(x_\ell, \mathrm{ne}(x_\ell)\big)\ f_2\big(\,\mathrm{ne}(x_\ell), x_m\big) = \frac{1}{Z''} p(x_\ell | \,\mathrm{ne}(x_\ell))\ p(x_m | \,\mathrm{ne}(x_\ell)) \end{aligned}$$

It follows that: $p(x_\ell, x_m | \,\mathrm{ne}(x_\ell)) = p(x_\ell | \,\mathrm{ne}(x_\ell))\ p(x_m | \,\mathrm{ne}(x_\ell)) \Leftrightarrow x_\ell \perp\!\!\!\perp x_m | \,\mathrm{ne}(x_\ell)$.

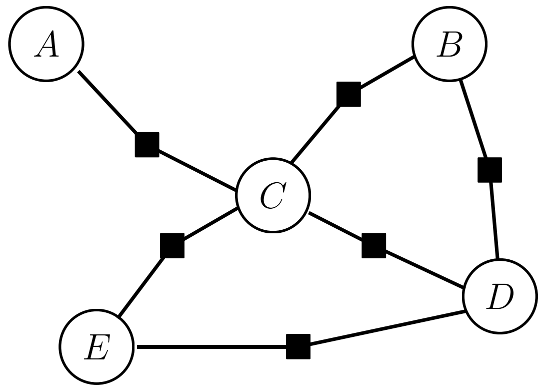
Comparing Undirected Graphs and Factor Graphs



(a)



(b)



(c)

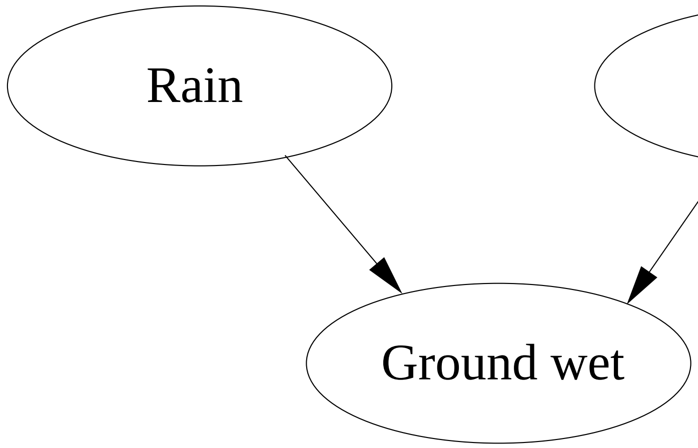
All nodes in (a), (b), and (c) have exactly the same neighbours and therefore these three graphs represent exactly the same conditional independence relationships.

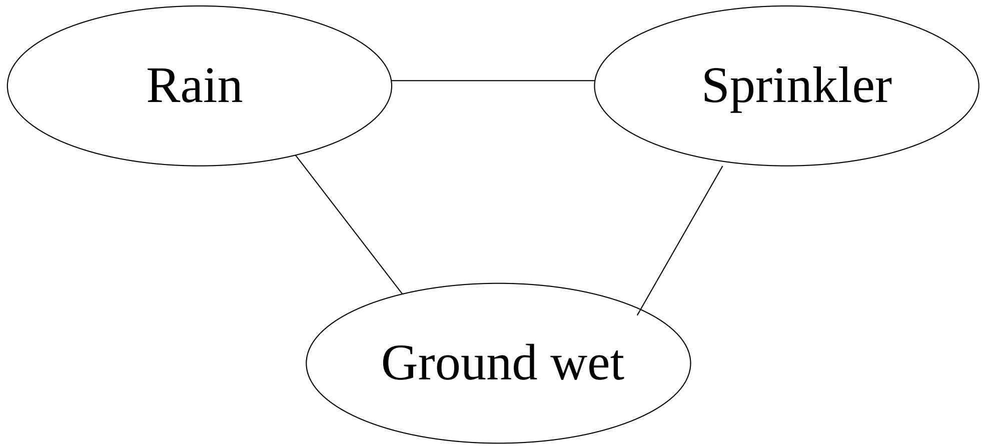
(c) also represents the fact that the probability factors into a product of pairwise functions.

Consider the case where each variables is 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 $\mathcal{O}(K^2)$.

Problems with Undirected Graphs and Factor Graphs

In UGs and FGs, many useful independencies are unrepresented—two variables are connected merely because some other variable depends on them:



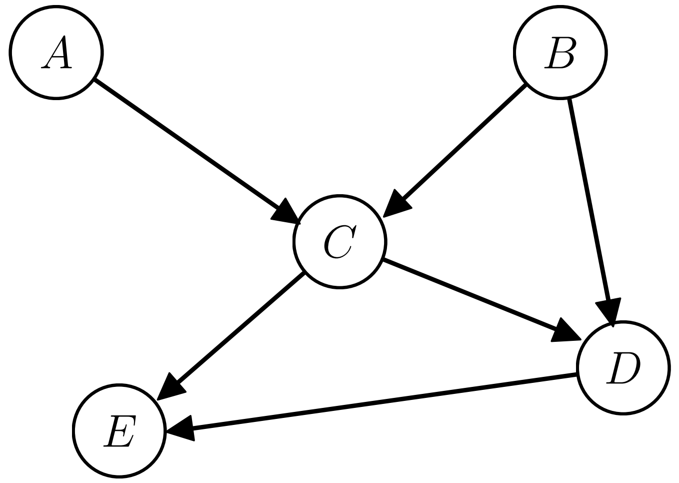


This highlights the difference between **marginal independence** and **conditional independence**.

R and S are marginally independent (i.e. given nothing), but they are conditionally dependent given G

"Explaining Away": Observing that the spinkler is on, explains away the fact that the ground was wet, therefore we don't need to believe that it rained.

Directed Acyclic Graphical Models (Bayes Net)



A DAG Model / Bayesian Network / Bayes Net represents a factorization of the joint probability distribution in terms of conditionals:

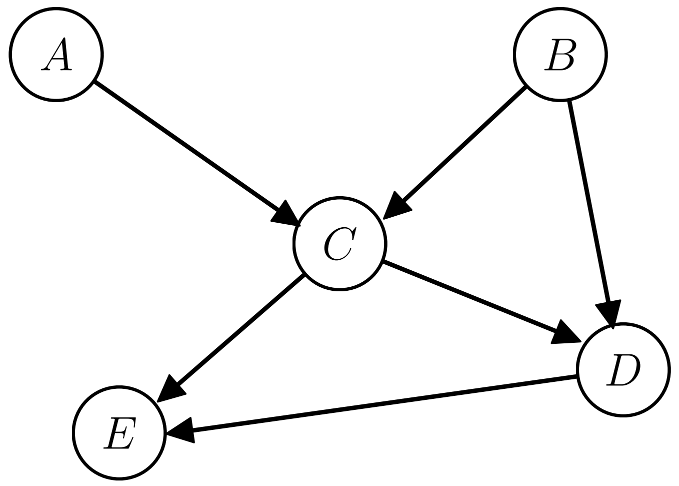
$$p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|B, C)p(E|C, D)$$

In general:

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

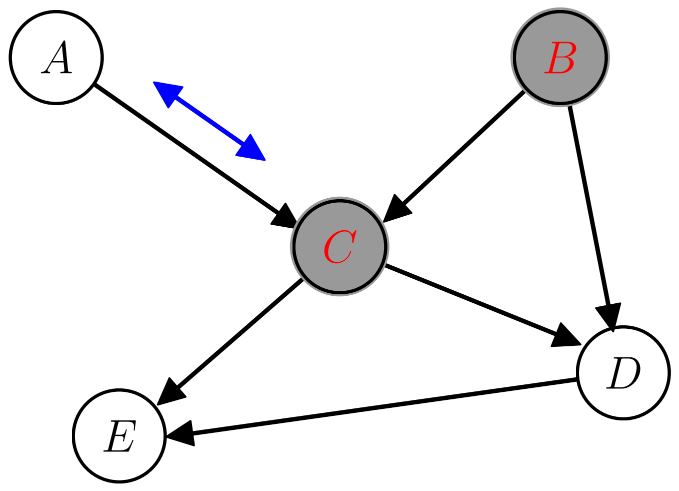
where pa(i) are the parents of node i.

Conditional Independence in Bayes Nets



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

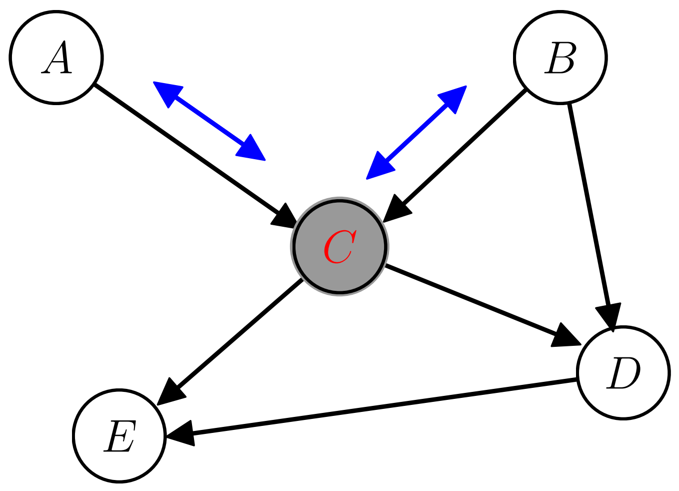
Conditional Independence in Bayes Nets



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

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

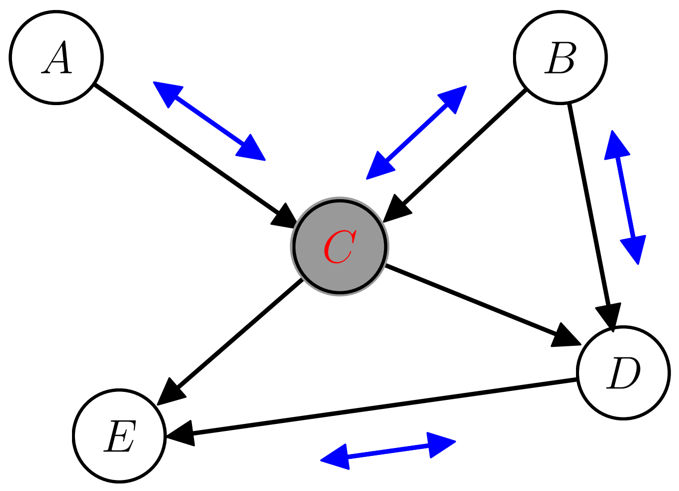
Conditional Independence in Bayes Nets



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

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

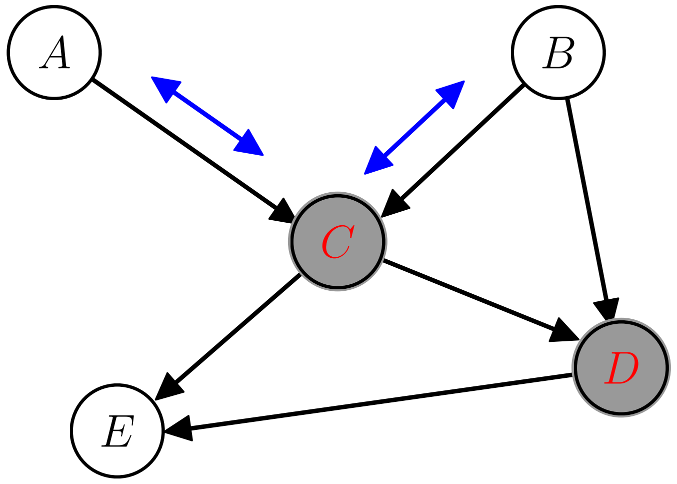
Conditional Independence in Bayes Nets



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

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

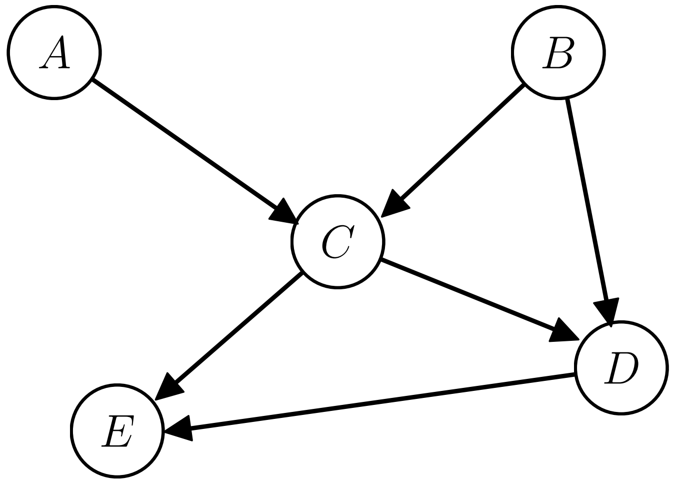
Conditional Independence in Bayes Nets



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

- $A \perp\!\!\!\perp E \mid \{B, C\}$: observed nodes block paths
- $A \not\!\perp\!\!\!\perp B \mid C$: observed node *creates* path by explaining away
- $A \not\!\perp\!\!\!\perp E \mid C$: created path extends to E via D
- $A \perp\!\!\!\perp E \mid \{C, D\}$: extra path blocked by observing D

Conditional Independence in Bayes Nets

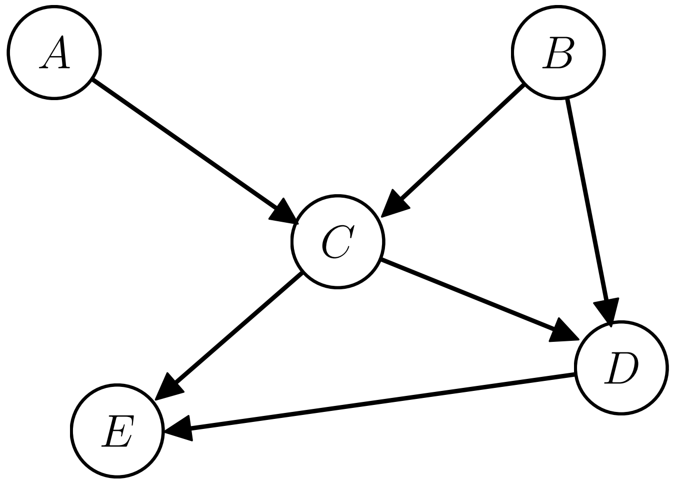


Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: observed nodes block paths
- $A \not\!\perp\!\!\!\perp B \mid C$: observed node *creates* path by explaining away
- $A \not\!\perp\!\!\!\perp E \mid C$: created path extends to E via D
- $A \perp\!\!\!\perp E \mid \{C, D\}$: extra path blocked by observing D

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

D-separation

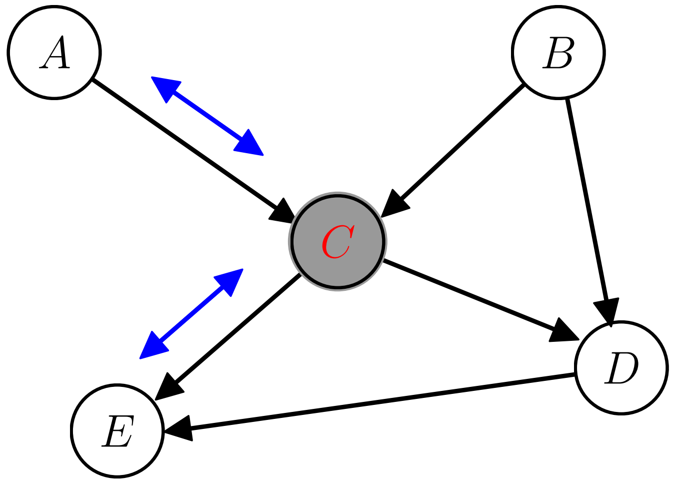


Consider two nodes X, Y and a set of observed nodes $\mathcal{V}$. When is $X \perp\!\!\!\perp Y | \mathcal{V}$?

We consider *every* undirected path[1] between X and Y.

[1]A path in the DAG ignoring the direction of edges. This is different to paths in the equivalent undirected graph.
[2]Note that converging arrows *along the path* only refers to what happens on that path.

D-separation



Consider two nodes X, Y and a set of observed nodes $\mathcal{V}$. When is $X \perp\!\!\!\perp Y | \mathcal{V}$?

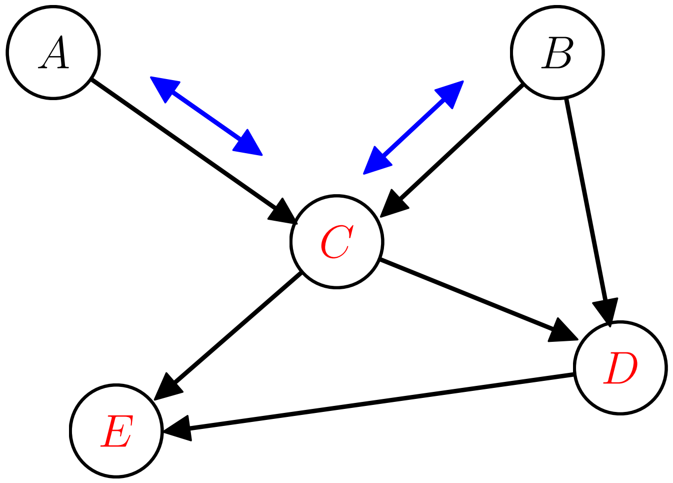
We consider *every* undirected path[1] between X and Y.

- The path is **blocked** if there is a non-collider node $W \in \mathcal{V}$ on the path ($\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow$). This is similar to the undirected graph semantics.

2

[1]A path in the DAG ignoring the direction of edges. This is different to paths in the equivalent undirected graph.
[2]Note that converging arrows *along the path* only refers to what happens on that path.

D-separation



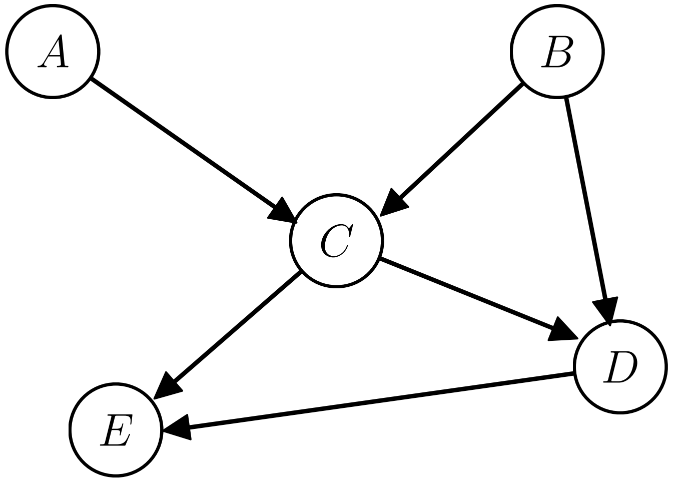
Consider two nodes X, Y and a set of observed nodes $\mathcal{V}$. When is $X \perp\!\!\!\perp Y | \mathcal{V}$?

We consider *every* undirected path[1] between X and Y.

- The path is **blocked** if there is a non-collider node $W \in \mathcal{V}$ on the path ($\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow$). This is similar to the undirected graph semantics.
- Alternatively, the path is **blocked** if there is a collider node W on the path ($\rightarrow W \leftarrow$)[2] such that **neither** W **nor any of its descendants are in** $\mathcal{V}$**.**

[1]A path in the DAG ignoring the direction of edges. This is different to paths in the equivalent undirected graph.
[2]Note that converging arrows *along the path* only refers to what happens on that path.

D-separation



Consider two nodes X, Y and a set of observed nodes $\mathcal{V}$. When is $X \perp\!\!\!\perp Y | \mathcal{V}$?

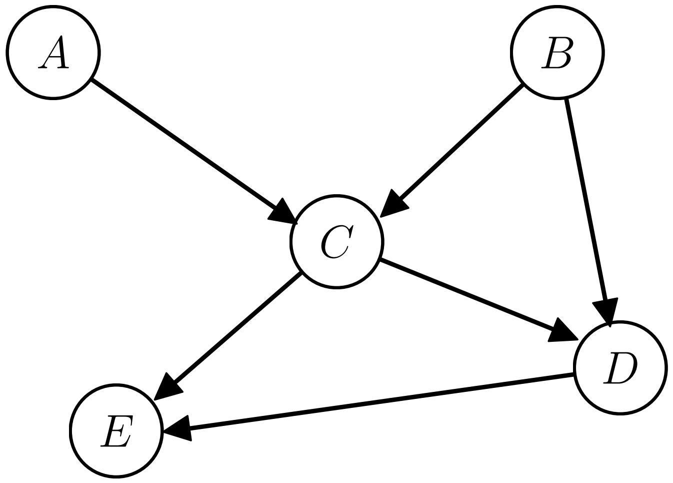
We consider *every* undirected path[1] between X and Y.

- The path is **blocked** if there is a non-collider node $W \in \mathcal{V}$ on the path ($\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow$). This is similar to the undirected graph semantics.
- Alternatively, the path is **blocked** if there is a collider node W on the path ($\rightarrow W \leftarrow$)[2] such that **neither W nor any of its descendants are in $\mathcal{V}$.**

Thus, if $\mathcal{V}$ contains at least one non-collider node or *no* collider nodes (or their descendants) along every path then $X \perp\!\!\!\perp Y | \mathcal{V}$. We say $\mathcal{V}$ d-separates X from Y (d for directed).

[1]A path in the DAG ignoring the direction of edges. This is different to paths in the equivalent undirected graph.
[2]Note that converging arrows *along the path* only refers to what happens on that path.

D-separation



Consider two nodes X, Y and a set of observed nodes $\mathcal{V}$. When is $X \perp\!\!\!\perp Y | \mathcal{V}$?

We consider *every* undirected path[1] between X and Y.

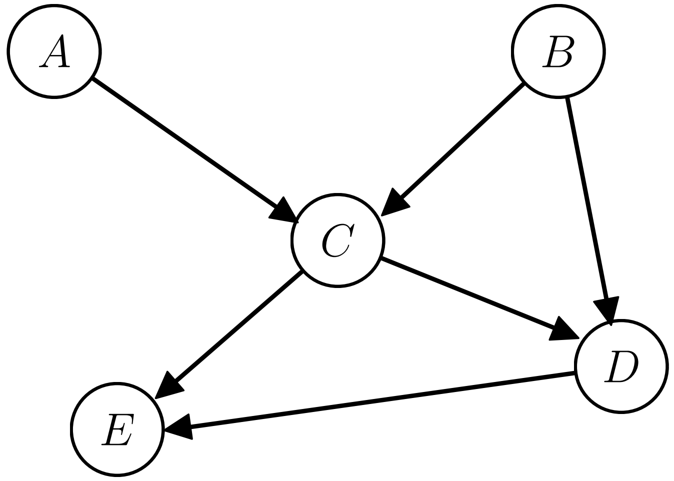
- The path is **blocked** if there is a non-collider node $W \in \mathcal{V}$ on the path ($\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow$). This is similar to the undirected graph semantics.
- Alternatively, the path is **blocked** if there is a collider node W on the path ($\rightarrow W \leftarrow$)[2] such that **neither** W **nor any of its descendants are in** $\mathcal{V}$**.**

Thus, if $\mathcal{V}$ contains at least one non-collider node or *no* collider nodes (or their descendants) along every path then $X \perp\!\!\!\perp Y | \mathcal{V}$. We say $\mathcal{V}$ d-separates X from Y (d for directed).

Corollary: Markov Boundary for X: $\{\text{parents}(X) \cup \text{children}(X) \cup \text{parents-of-children}(X)\}$.

[1] A path in the DAG ignoring the direction of edges. This is different to paths in the equivalent undirected graph.
[2] Note that converging arrows *along the path* only refers to what happens on that path.

The "Bayes-ball" algorithm

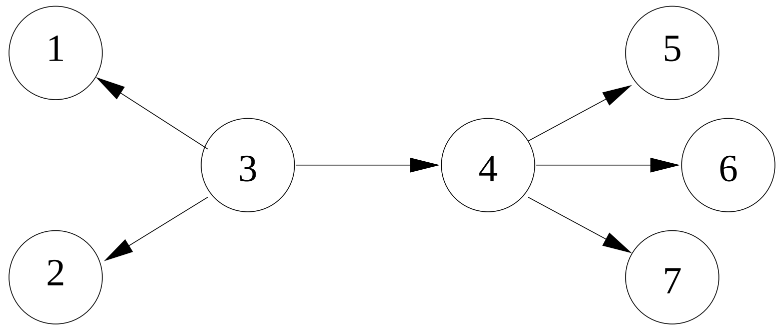


Game: can you get a ball from X to Y without being blocked by $\mathcal{V}$? If so, $X \not\perp\!\!\!\perp Y | \mathcal{V}$

Rules: Depending on the direction the ball came from and the type of node, the ball can **pass through** (from a parent to all children, from a child to all parents), **bounce back** (from any parent to all parents, or from any child to all children), or be **blocked**.

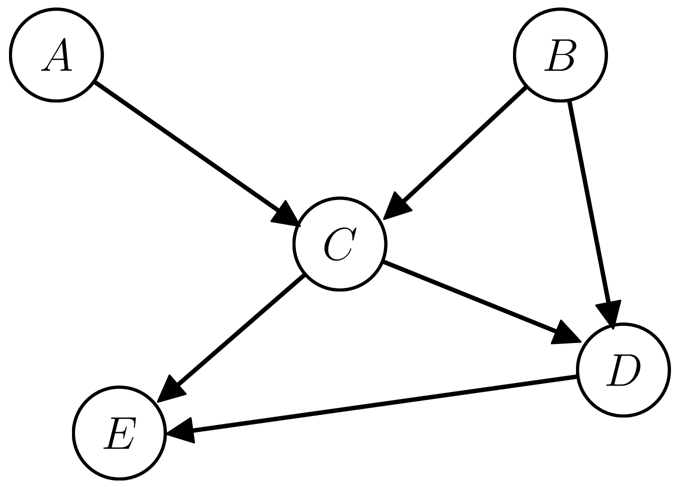
- An unobserved (hidden) node ($W \notin \mathcal{V}$) passes balls through but also bounces back balls from children.
- An observed (given) node ($W \in \mathcal{V}$) bounces back balls from parents but blocks balls from children.

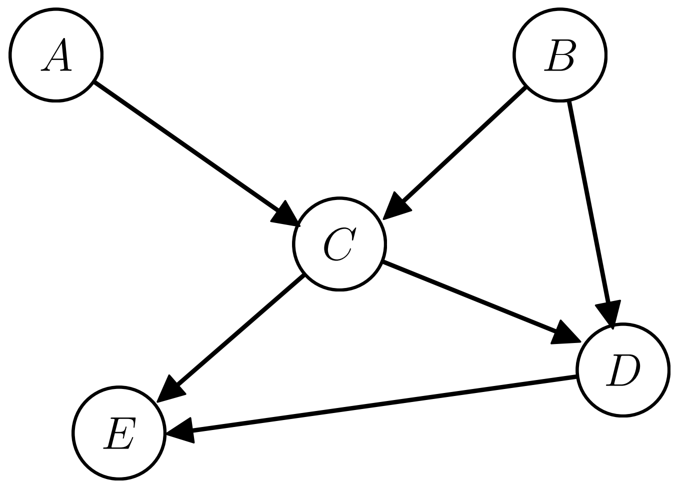
From Directed Trees to Undirected Trees



$$
\begin{aligned}
p(x_1, x_2, \ldots, x_7) &= p(x_3)p(x_1|x_3)p(x_2|x_3)p(x_4|x_3)p(x_5|x_4)p(x_6|x_4)p(x_7|x_4) \\
&= \frac{p(x_1,x_3)p(x_2,x_3)p(x_3,x_4)p(x_4,x_5)p(x_4,x_6)p(x_4,x_7)}{p(x_3)p(x_3)p(x_4)p(x_4)p(x_4)} \\
&= \frac{\text{product of cliques}}{\text{product of clique intersections}} \\
&= g_1(x_1,x_3)g_2(x_2,x_3)g_3(x_3,x_4)g_4(x_4,x_5)g_5(x_4,x_6)g_6(x_4,x_7) = \\
&= \prod_i g_i(C_i)
\end{aligned}
$$

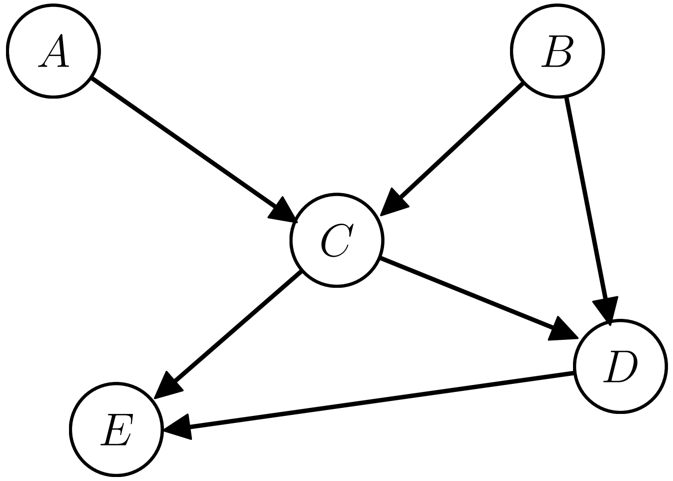
From DAGs to Factor (and Undirected) Graphs

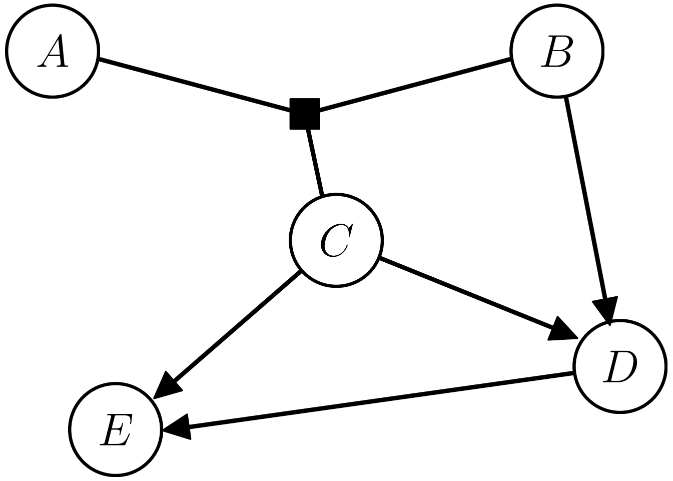




$$P(ABCDE) = P(A)P(B)P(C|AB)P(D|BC)P(E|CD)$$

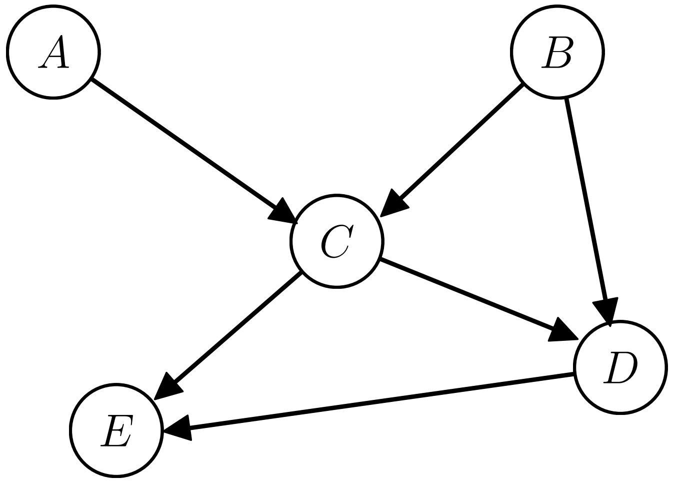
From DAGs to Factor (and Undirected) Graphs

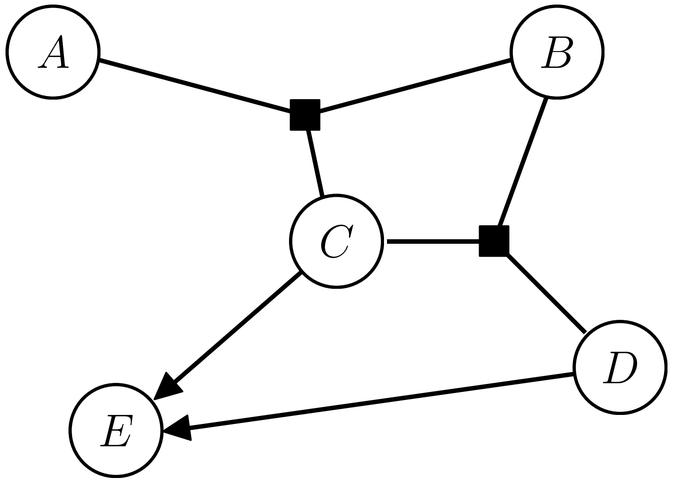




$$\begin{aligned} P(ABCDE) &= \underbrace{P(A)P(B)P(C|AB)} P(D|BC)P(E|CD) \\ &= \qquad g(ABC) \qquad P(D|BC)P(E|CD) \end{aligned}$$

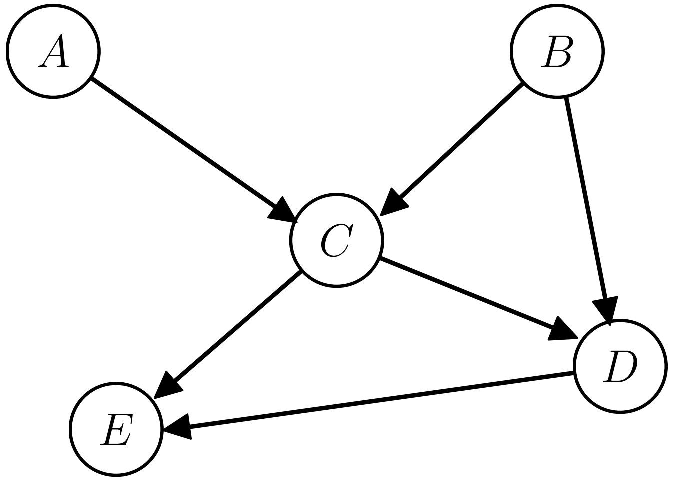
From DAGs to Factor (and Undirected) Graphs

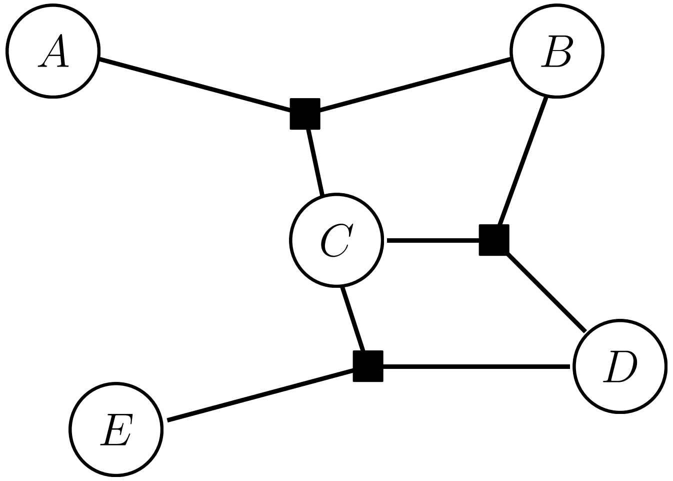




$$
\begin{aligned}
P(ABCDE) &= \underbrace{P(A)P(B)P(C|AB)} P(D|BC)P(E|CD) \\
&= \qquad g(ABC) \qquad P(D|BC)P(E|CD) \\
&= \qquad g(ABC) \qquad g(DBC) \;\; P(E|CD)
\end{aligned}
$$

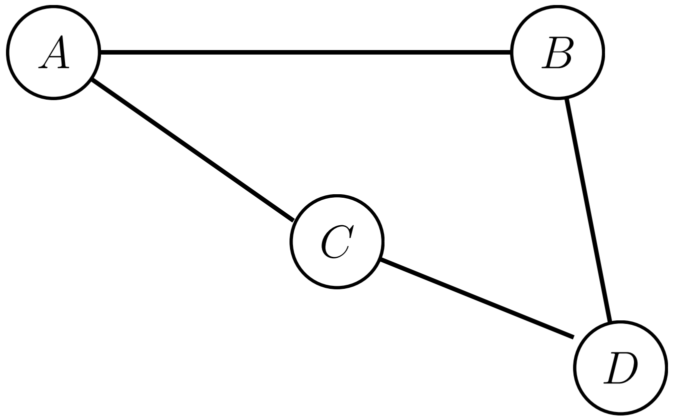
From DAGs to Factor (and Undirected) Graphs





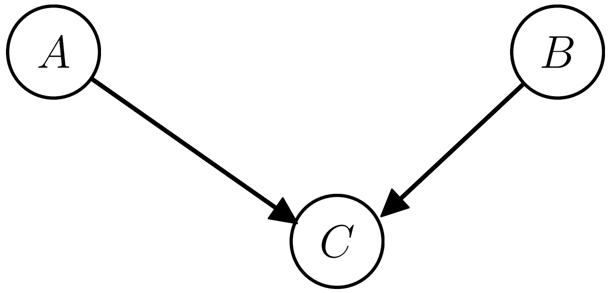
$$\begin{aligned}
P(ABCDE) &= \underbrace{P(A)P(B)P(C|AB)}\,P(D|BC)P(E|CD) \\
&= \quad g(ABC) \quad P(D|BC)P(E|CD) \\
&= \quad g(ABC) \quad g(DBC) \;\; P(E|CD) \\
&= \quad g(ABC) \quad g(DBC) \;\; g(ECD)
\end{aligned}$$

Expressive Power of Directed and Undirected Graphs



No Directed Graph (Bayes Net) 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 $\Longrightarrow$ dependence in Directed Graph but independence in Undirected Graph.



No Undirected Graph or Factor Graph can represent these and only these independencies

Message Passing

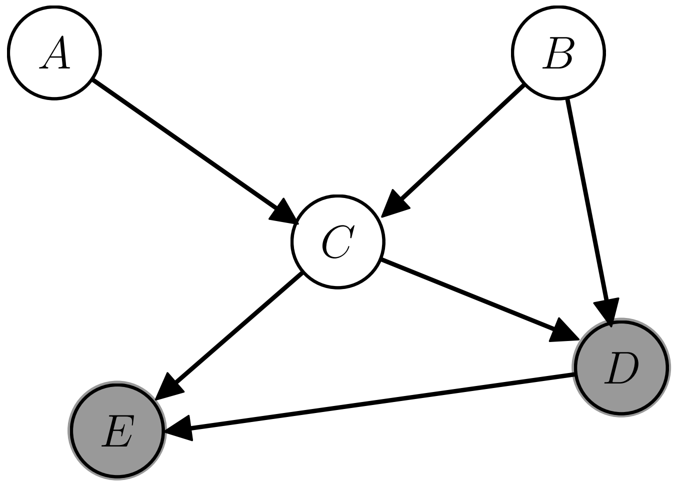
Thus far, we have used graphical models to encode the conditional independences of probability distributions visually. Can they do more?

We often need to compute a function of the distribution on hidden nodes conditioned on some observed ones.

- marginals: $P(A|DE), \ldots$.
- most likely values: argmax $P(ABC|DE)$

Message passing algorithms exploit conditional independence relationships to make this computation efficient. Applies to any distributive function.

- forward-backward
- Viterbi



In the general case, these algorithms are defined by message-passing rules on the appropriate graph.

Learning

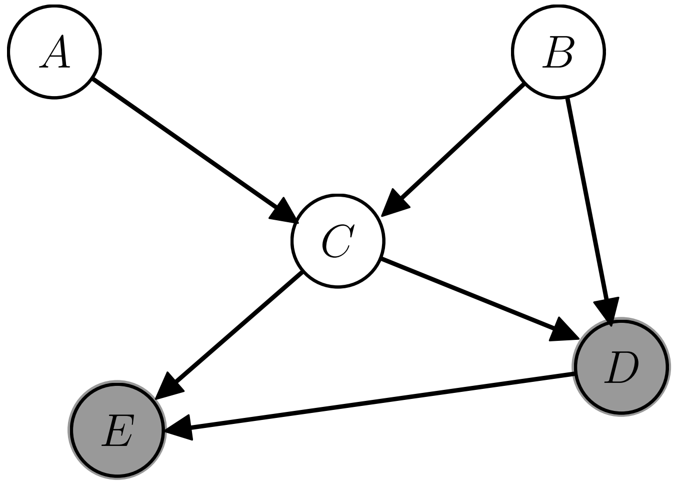
In combination with an appropriate message passing algorithm, the factored structure implied by the graph also makes learning easy.

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

$$\begin{aligned}\mathcal{L} &= P(\textsf{obs}|\theta)\\ &= \int P(\textsf{obs}, \textsf{unobs}|\theta)\; d(\textsf{unobs})\end{aligned}$$

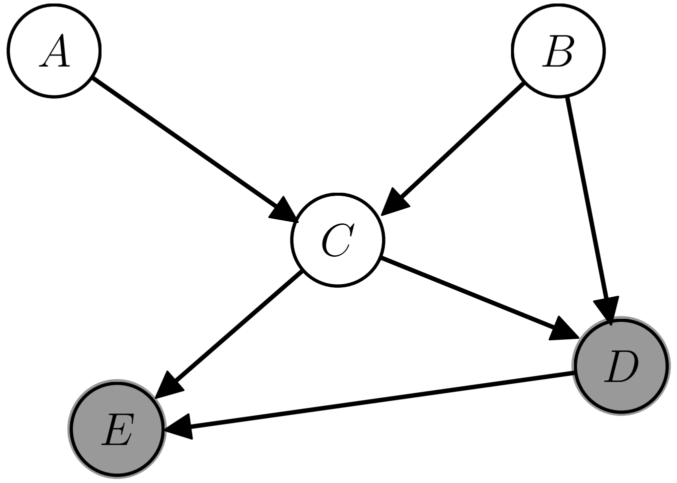
by EM, need to maximise

$$\begin{aligned}\mathcal{F} &= \big\langle \log P(\textsf{obs}, \textsf{unobs}|\theta)\big\rangle_{P(\textsf{unobs}|\textsf{obs})}\\ &= \left\langle \sum_i \log g(C_i|\theta_i) - \log Z \right\rangle_{P(\textsf{unobs}|\textsf{obs})}\\ &= \sum_i \big\langle \log g(C_i|\theta_i)\big\rangle_{P(C_i\backslash\textsf{obs}|\textsf{obs})} - \log Z\end{aligned}$$



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

Finding marginals by message passing



We will discuss algorithms to compute single node marginals (as in the forward-backward algorithm). The same scheme is easily adapted to clique-marginals or other functions.

Goal: For a node X and evidence e, compute $p(X|e)$. [3]

We will consider a number of cases in turn:

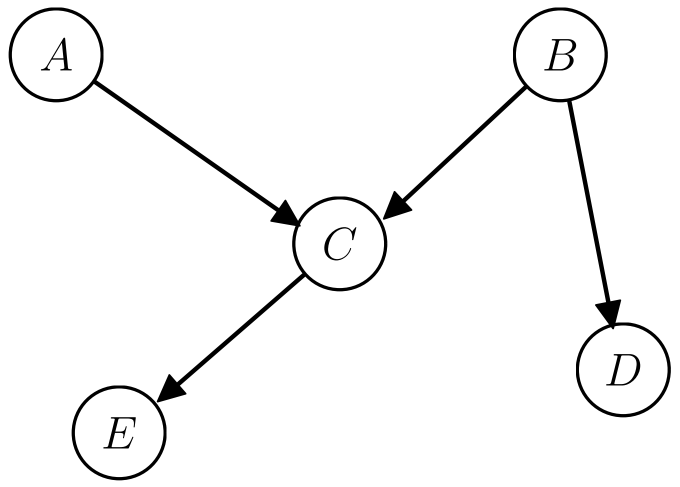
- Singly-connected DAGs
- Markov trees
- Arbitrary Markov (and Factor) graphs
- Arbitrary DAGs

[3] *i.e.* observed values for some nodes. This usage of "evidence" has nothing to do with the marginal likelihood.

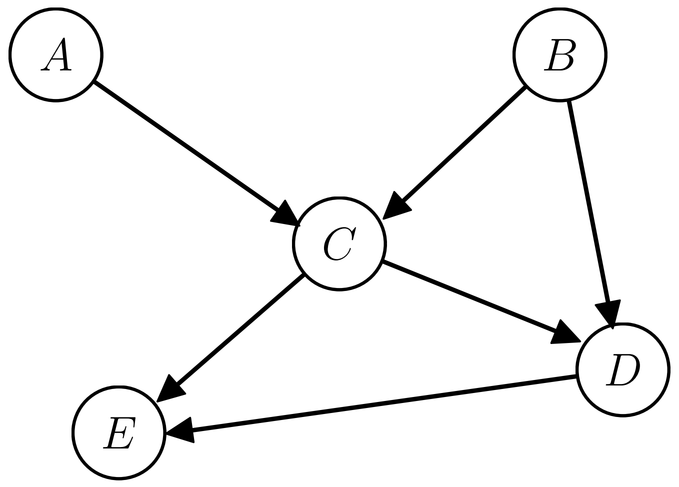
Case I: Singly Connected DAGs

Message passing algorithms are convergent (and exact) when executed on trees (*i.e.* graphs with no loops). Easy to avoid circular message paths.

A single-connected Bayesian network (SCBN) is a DAG where the graph of all undirected paths forms a tree, *i.e.*, there is only one undirected path between any two nodes. This does not mean that the *equivalent* Markov graph is a tree.



SINGLY-CONNECTED



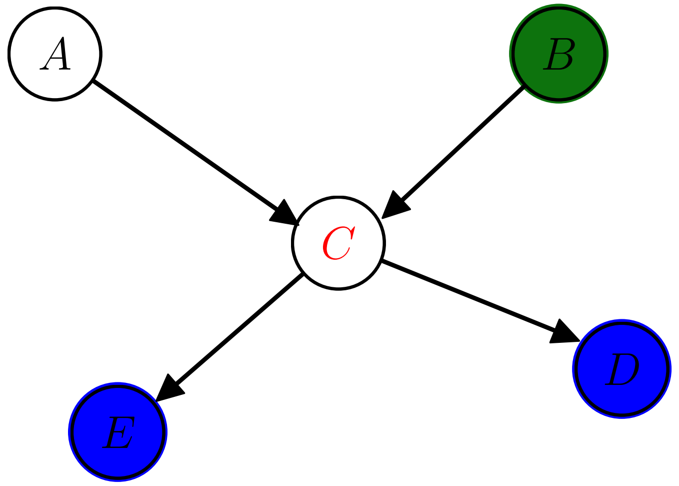
LOOPY

Less restrictive than requiring that DAG be a tree (*i.e.* that all nodes have no more than 1 parent), but sufficient for message passing to converge.

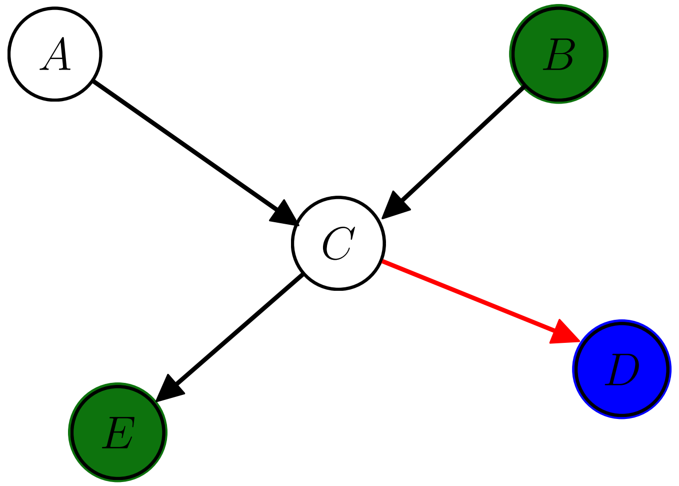
Upstream and Downstream Evidence

In an SCBN:

- every node X divides the evidence into upstream e_X^+ and downstream e_X^-.



- every edge $X \rightarrow Y$ divides the evidence into upstream e_{XY}^+ and downstream e_{XY}^-.



Belief Propagation

The algorithm to find marginal distributions in a SCBN is called belief propagation (BP). It relies on three key ideas (c.f. the α and β messages from FB):

Idea 1: Beliefs about X can be found by combining upstream and downstream evidence:

$$p(X|e) = \frac{p(X,e)}{p(e)} = \frac{p(X,e_X^+,e_X^-)}{p(e_X^+,e_X^-)} \propto p(X|e_X^+) \times \underbrace{p(e_X^-|X,e_X^+)}_{X \text{ d-separates } e_X^- \text{ from } e_X^+}$$

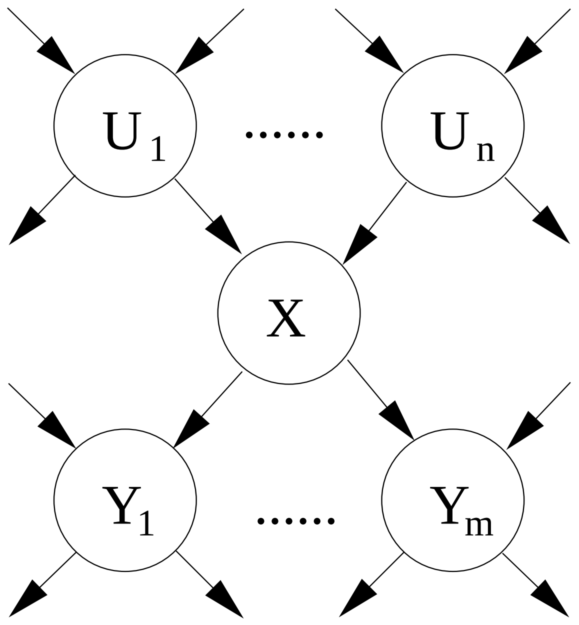
$$= p(X|e_X^+)p(e_X^-|X) = \pi(X)\lambda(X)$$

If we think of X as a parameter, $\pi(X)$ resembles its prior and $\lambda(X)$ resembles a likelihood.

Idea 2: The upstream and downstream evidence can be computed recursively via a local message passing algorithm between the nodes in the graph.

Idea 3: “Don’t send back to a node (any part of) the message it sent to you!”

Belief Propagation



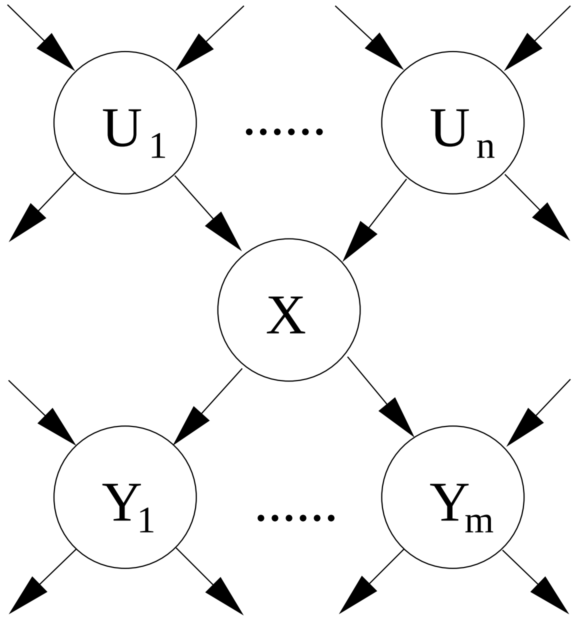
top-down causal support:
$\pi_X(U_i) = p(U_i | e^+_{U_i X})$

bottom-up diagnostic support:
$\lambda_{Y_j}(X) = p(e^-_{XY_j} | X)$

To update the belief about X:

$$\begin{aligned}
\mathrm{BEL}(X) &= \frac{1}{Z}\lambda(X)\pi(X) \\
\lambda(X) &= \prod_j \lambda_{Y_j}(X) \\
\pi(X) &= \sum_{U_1 \cdots U_n} p(X | U_1, \ldots, U_n) \prod_i \pi_X(U_i)
\end{aligned}$$

Belief Propagation



top-down causal support:
$\pi_X(U_i) = p(U_i | e^+_{U_i X})$

bottom-up diagnostic support:
$\lambda_{Y_j}(X) = p(e^-_{XY_j} | X)$

Bottom-up propagation, message X sends to U_i:

$$\lambda_X(U_i) \;=\; \sum_X \lambda(X) \sum_{U_k : k \neq i} p(X | U_1, \ldots, U_n) \prod_{k \neq i} \pi_X(U_k)$$

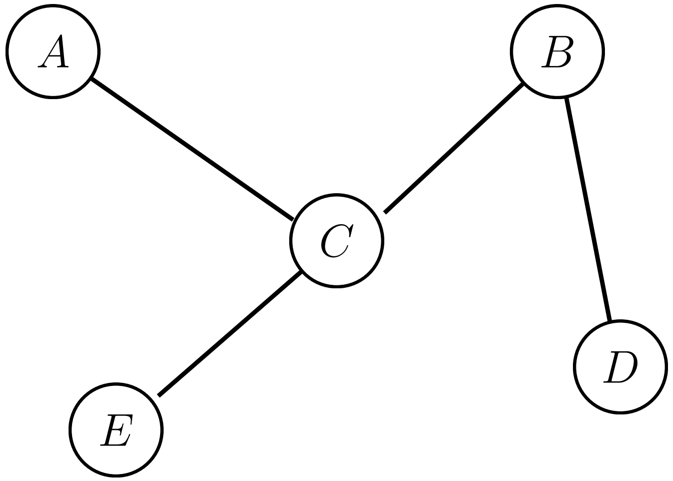
Top-down propagation, message X sends to Y_j:

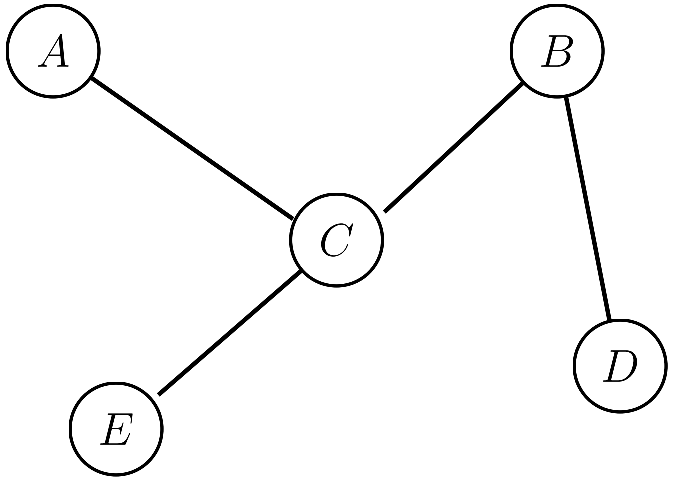
$$\pi_{Y_j}(X) \;=\; \frac{1}{Z} [\prod_{k \neq j} \lambda_{Y_k}(X)] \sum_{U_1 \cdots U_n} p(X | U_1, \ldots, U_n) \prod_i \pi_X(U_i) \;=\; \frac{1}{Z} \frac{\mathrm{BEL}(X)}{\lambda_{Y_j}(X)}$$

Z is the normaliser ensuring $\sum_X \pi_{Y_j}(X) = 1$

Case II: Markov Trees

Any Markov tree can be turned into an SCBN by arbitrarily choosing a root node, and directing all edges away from it.

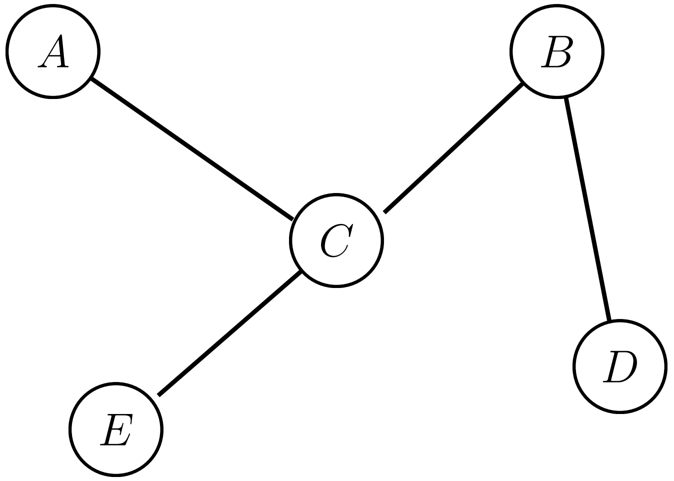


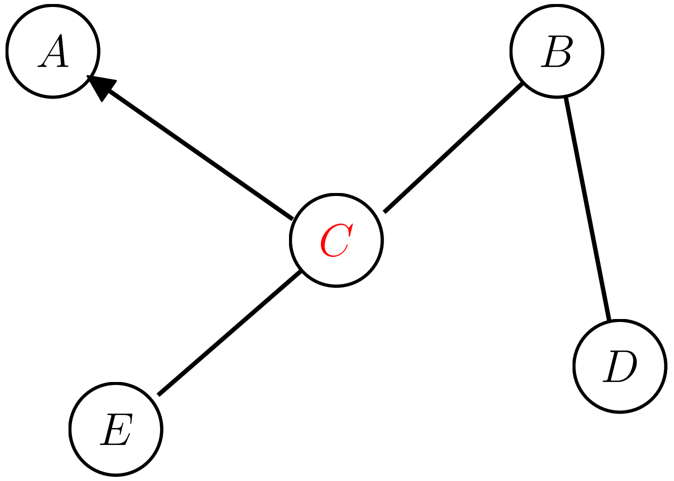


$$P(ABCDE) \propto g_1(AC)g_2(BC)g_3(CE)g_4(BD)$$

Case II: Markov Trees

Any Markov tree can be turned into an SCBN by arbitrarily choosing a root node, and directing all edges away from it.



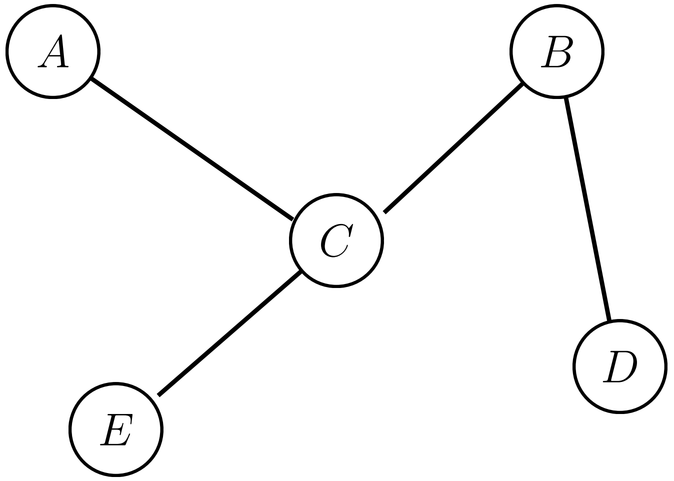


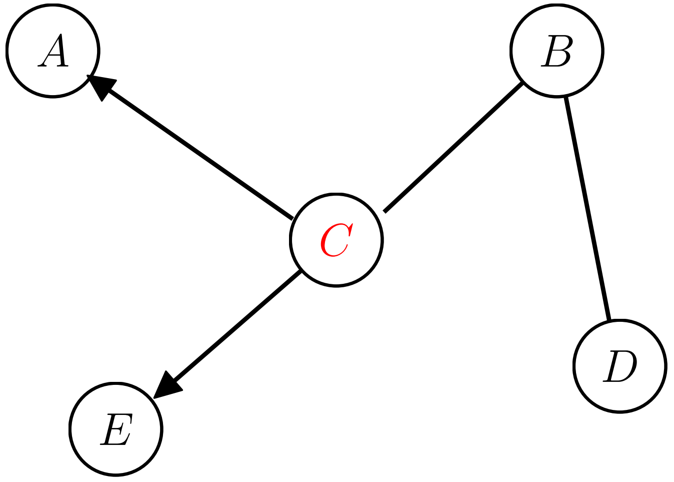
$$A \perp\!\!\!\perp BDE \mid C$$

$$\begin{aligned} &P(ABCDE) \\ &\quad = P(C)P(A|C)P(BDE|C) \end{aligned}$$

Case II: Markov Trees

Any Markov tree can be turned into an SCBN by arbitrarily choosing a root node, and directing all edges away from it.



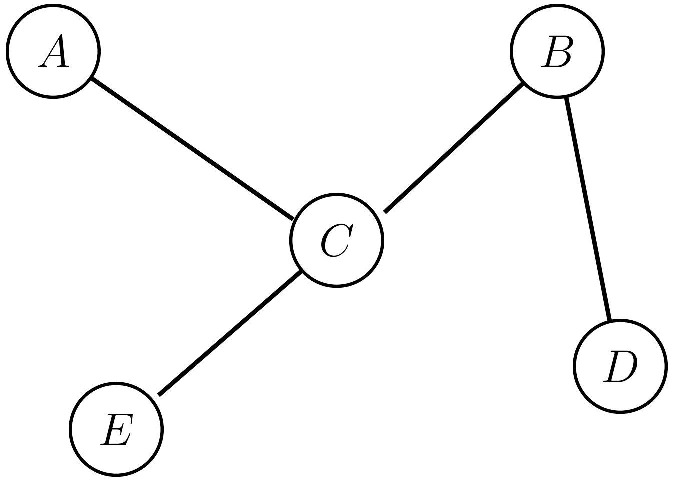


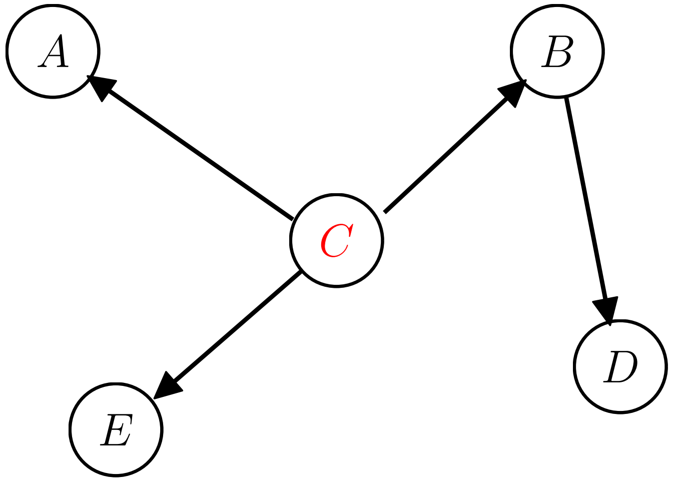
$$A \perp\!\!\!\perp BDE \mid C$$
$$E \perp\!\!\!\perp BD \mid C$$

$$\begin{aligned} &P(ABCDE) \\ &\quad = P(C)P(A|C)P(BDE|C) \\ &\quad = P(C)P(A|C)P(E|C)P(BD|C) \end{aligned}$$

Case II: Markov Trees

Any Markov tree can be turned into an SCBN by arbitrarily choosing a root node, and directing all edges away from it.



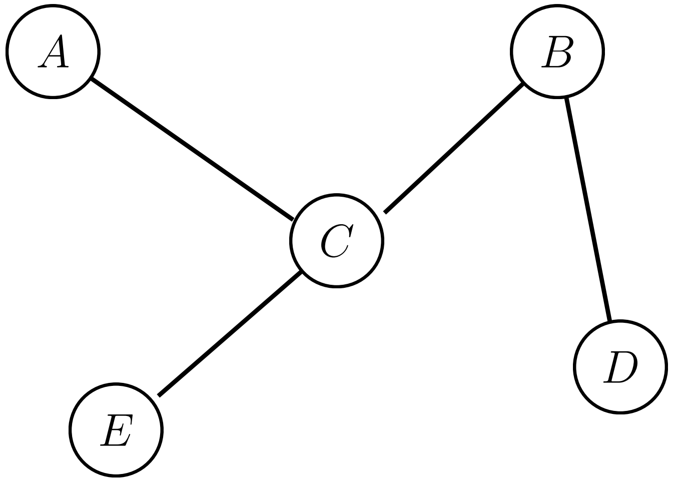


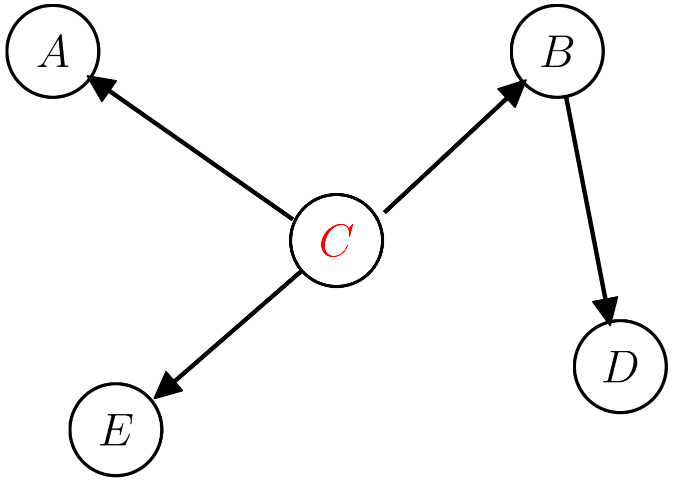
$$P(ABCDE)$$

$$\begin{array}{cl}
A \perp\!\!\!\perp BDE \mid C & = P(C)P(A|C)P(BDE|C) \\
E \perp\!\!\!\perp BD \mid C & = P(C)P(A|C)P(E|C)P(BD|C) \\
D \perp\!\!\!\perp C \mid B & = P(C)P(A|C)P(E|C)P(B|C)P(D|B\cancel{C})
\end{array}$$

Case II: Markov Trees

Any Markov tree can be turned into an SCBN by arbitrarily choosing a root node, and directing all edges away from it.



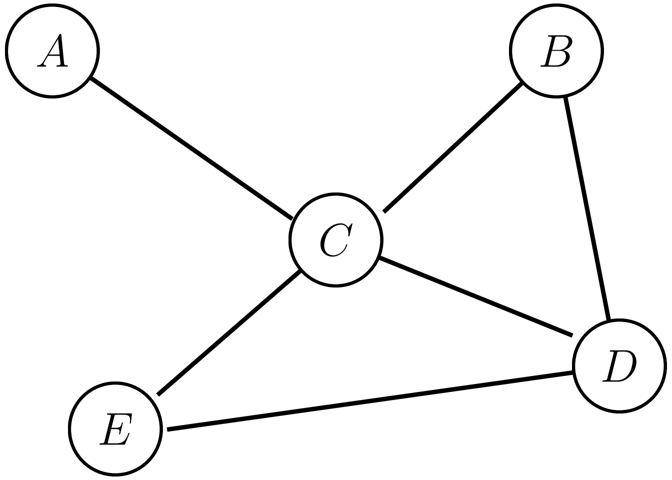


$$\begin{array}{ll}
 & P(ABCDE) \\
A \perp\!\!\!\perp BDE \mid C & \quad = P(C)P(A|C)P(BDE|C) \\
E \perp\!\!\!\perp BD \mid C & \quad = P(C)P(A|C)P(E|C)P(BD|C) \\
D \perp\!\!\!\perp C \mid B & \quad = P(C)P(A|C)P(E|C)P(B|C)P(D|B\cancel{C})
\end{array}$$

This DAG will always be a tree (no colliders), and therefore encodes exactly the same conditional independencies (and distribution) as the Markov tree.

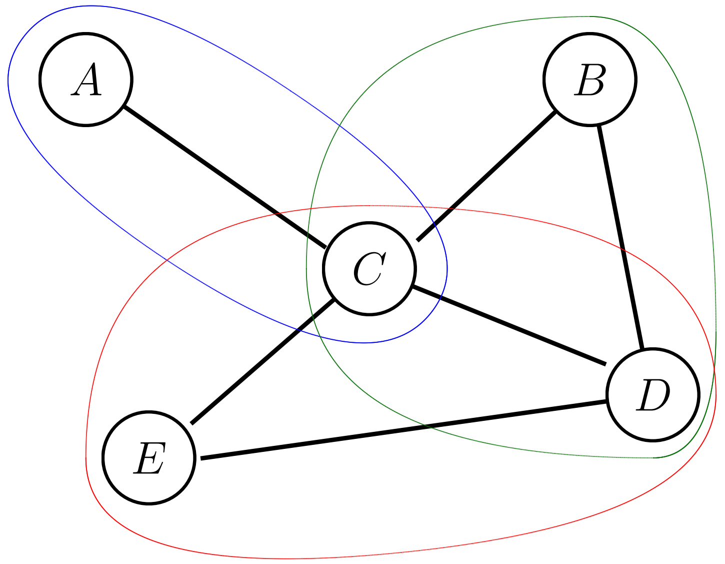
Thus, belief propagation in the DAG will find the correct marginals for the undirected graph.

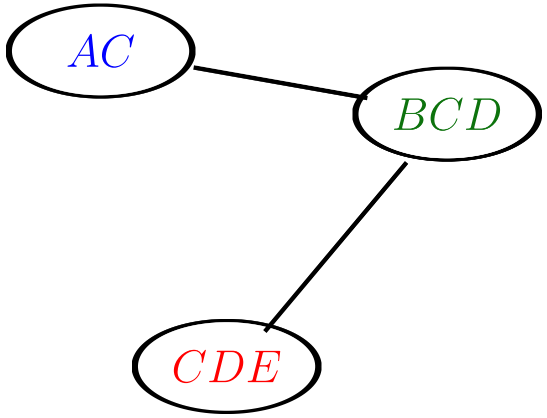
Case III: Markov Networks



In a general Markov network (undirected graph), loops may prevent BP from converging.

Case III: Markov Networks

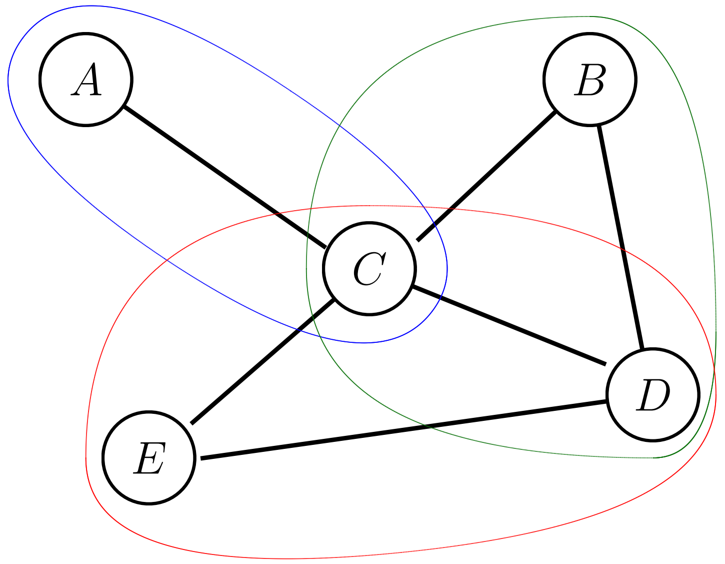


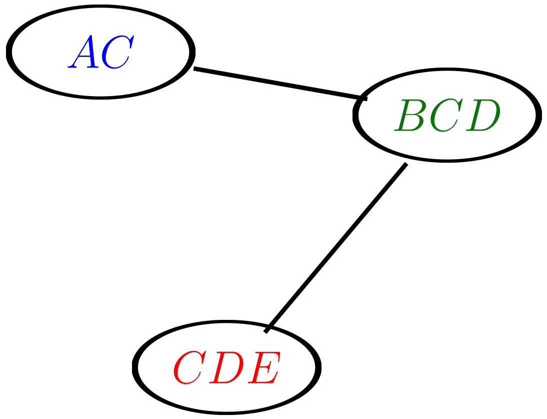


In a general Markov network (undirected graph), loops may prevent BP from converging.

Instead, construct the join or junction tree from the (maximal) cliques of the graph. Here $AC \perp\!\!\!\perp CDE \mid BCD$ because $A \perp\!\!\!\perp E \mid \{BCD\}$ and C and D are fixed by conditioning.

Case III: Markov Networks



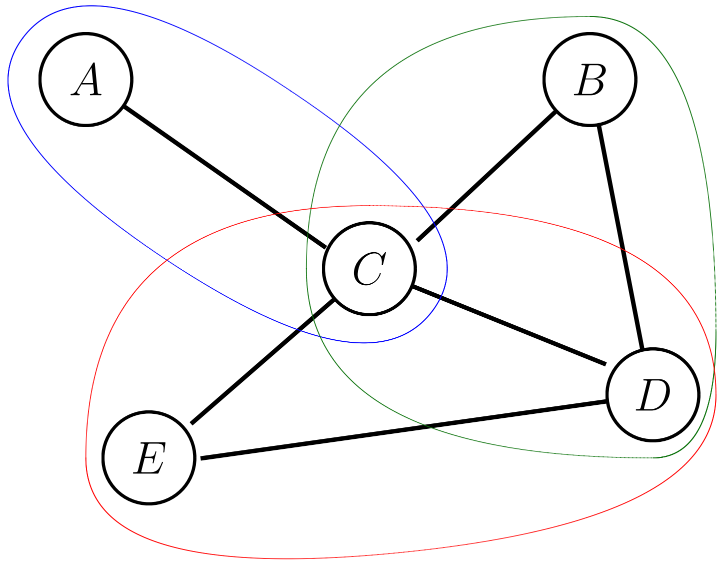


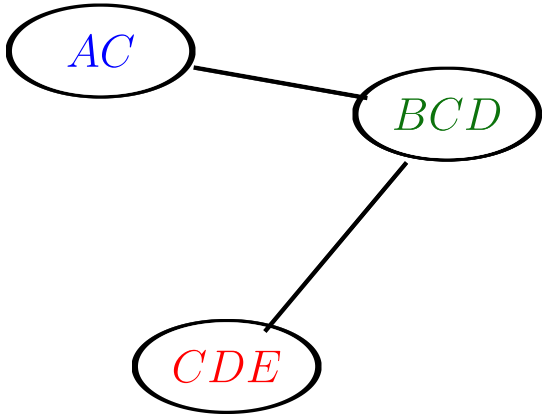
In a general Markov network (undirected graph), loops may prevent BP from converging.

Instead, construct the join or junction tree from the (maximal) cliques of the graph. Here $AC \perp\!\!\!\perp CDE \mid BCD$ because $A \perp\!\!\!\perp E \mid \{BCD\}$ and C and D are fixed by conditioning.

Now, carry out belief propagation in the join tree. Messages turn out to be distributions over nodes shared by cliques (called separators). This is the Junction-tree algorithm

Case III: Markov Networks





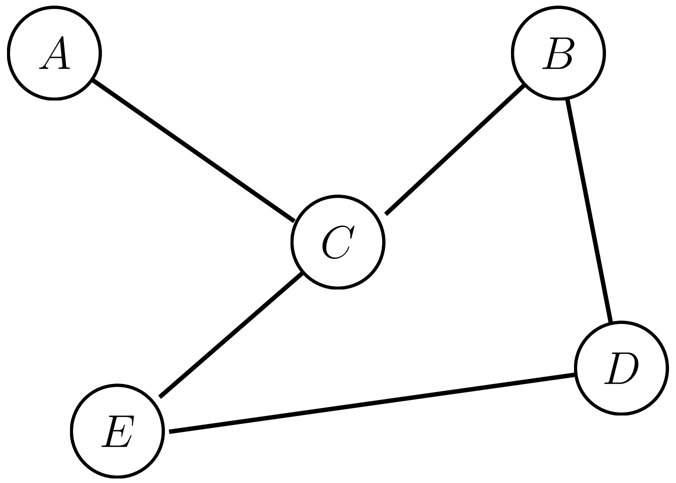
In a general Markov network (undirected graph), loops may prevent BP from converging.

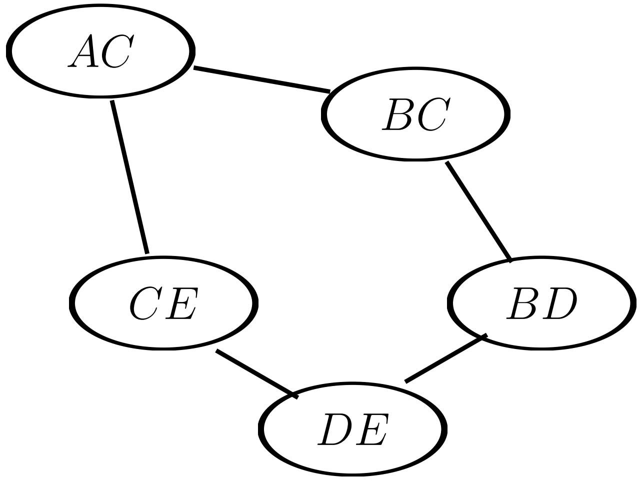
Instead, construct the join or junction tree from the (maximal) cliques of the graph. Here $AC \perp\!\!\!\perp CDE \mid BCD$ because $A \perp\!\!\!\perp E \mid \{BCD\}$ and C and D are fixed by conditioning.

Now, carry out belief propagation in the join tree. Messages turn out to be distributions over nodes shared by cliques (called separators). This is the Junction-tree algorithm

This yields the clique-marginals, which might be what we actually want (*e.g.* for learning). Otherwise, marginalise to obtain distributions on single nodes.

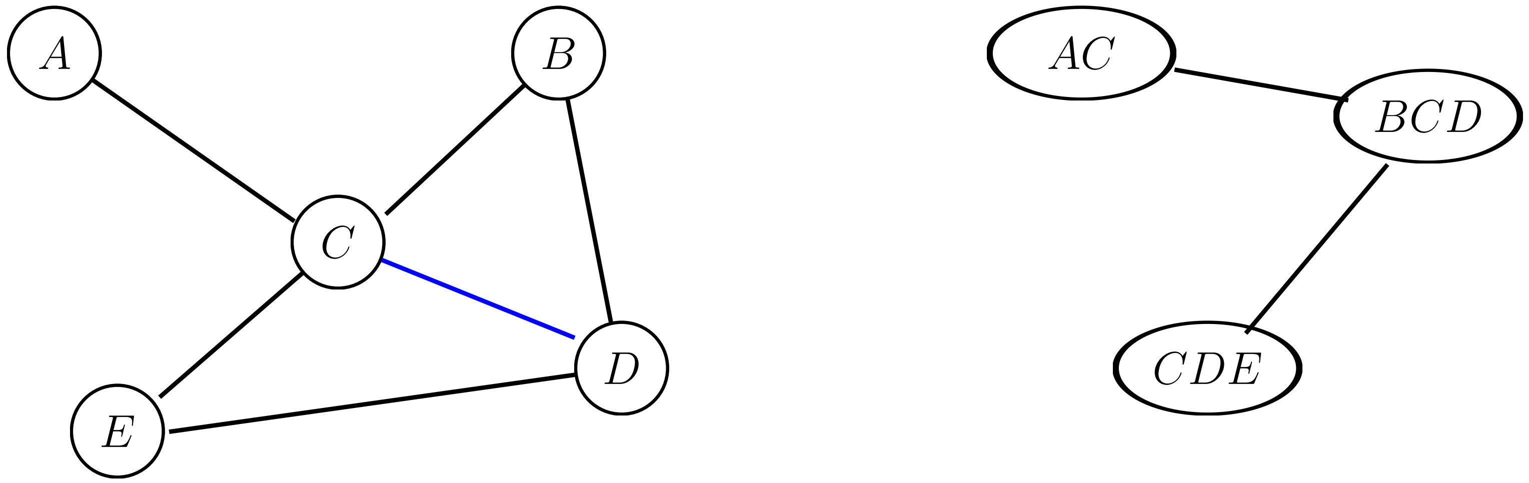
Triangulation





If the graph contains loops of more than 3 nodes, the cliques will not form a tree. (Why is the loop length here 5, not 4?)

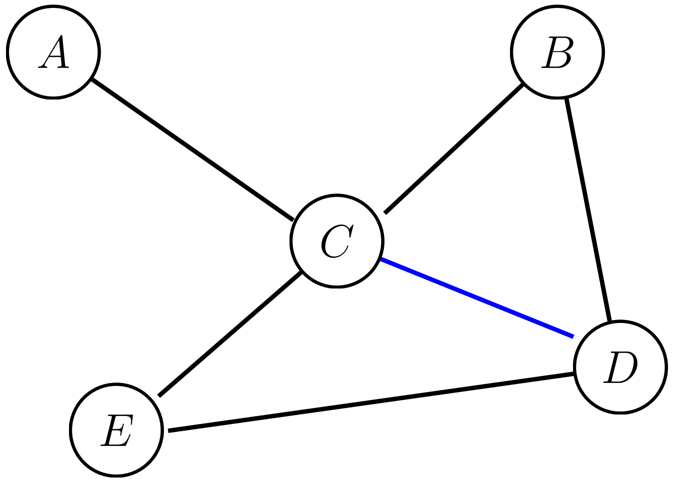
Triangulation

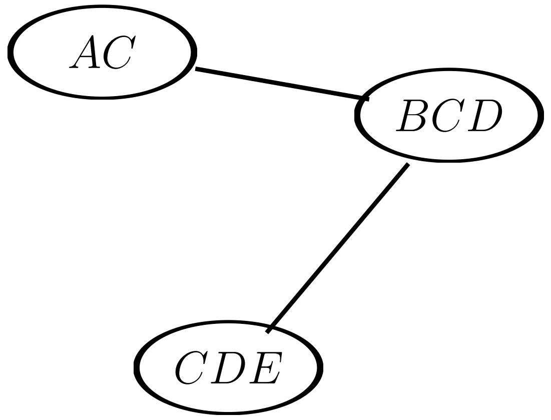


If the graph contains loops of more than 3 nodes, the cliques will not form a tree. (Why is the loop length here 5, not 4?)

To obtain a tree we need to introduce an additional edge to the original loop. This is called triangulation — place chords within each (>3)-loop so as to reduce them all to triangles. We give up some knowledge (of conditional independence) in exchange for a tractable algorithm.

Triangulation





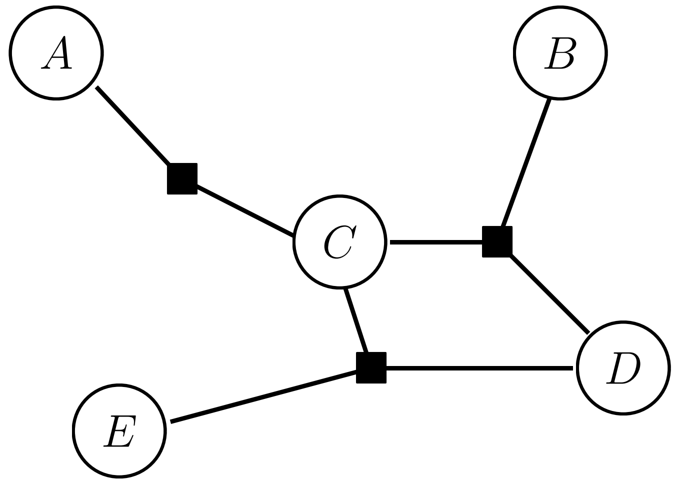
If the graph contains loops of more than 3 nodes, the cliques will not form a tree. (Why is the loop length here 5, not 4?)

To obtain a tree we need to introduce an additional edge to the original loop. This is called triangulation — place chords within each (>3)-loop so as to reduce them all to triangles. We give up some knowledge (of conditional independence) in exchange for a tractable algorithm.

There are many ways to do this. In this case either $\overline{CD}$ or $\overline{BE}$ would have worked equally well. In general, though, finding the best triangulation of a graph (smallest cliques, most efficient BP) is NP-hard.

Factor Graphs

One way to see how message passing on a Junction tree works is to consider the corresponding Factor graph of cliques. To implement BP in the factor graph setting, we consider messages that pass between variables and the *factors*.



Consider a vector of variables $\mathbf{x} = (x_1, \ldots, x_n)$, such that

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_j f_j(\mathbf{x}_{C_j})$$

C_j denotes the subset of $\{1, \ldots, n\}$ which participate in factor f_j and $\mathbf{x}_{C_j} = \{x_i : i \in C_j\}$.

(We will assume our factor graph is a tree. If not, it may need triangulation.)

Propagation in Factor Graphs

We can compute probabilities in a factor graph by propagating messages from variable nodes to function nodes and *vice versa*.

Propagation in Factor Graphs

We can compute probabilities in a factor graph by propagating messages from variable nodes to function nodes and *vice versa*.

$$\mathrm{ne}(x_i) \overset{\mathrm{def}}{=} \{f_j : x \in C_j\} \qquad \mathrm{ne}(f_j) \overset{\mathrm{def}}{=} \{x_i : x_i \in C_j\}$$

Propagation in Factor Graphs

We can compute probabilities in a factor graph by propagating messages from variable nodes to function nodes and *vice versa*.

$$\mathrm{ne}(x_i) \overset{\mathrm{def}}{=} \{f_j : x \in C_j\} \qquad \mathrm{ne}(f_j) \overset{\mathrm{def}}{=} \{x_i : x_i \in C_j\}$$

variable $x \rightarrow$ **factor** f:

$$\mu_{x\to f}(x) = \prod_{h \in \mathrm{ne}(x) \setminus \{f\}} \mu_{h \to x}(x)$$

Propagation in Factor Graphs

We can compute probabilities in a factor graph by propagating messages from variable nodes to function nodes and *vice versa*.

$$\mathrm{ne}(x_i) \overset{\mathrm{def}}{=} \{f_j : x \in C_j\} \qquad \mathrm{ne}(f_j) \overset{\mathrm{def}}{=} \{x_i : x_i \in C_j\}$$

variable $x \rightarrow$ **factor** f:

$$\mu_{x\rightarrow f}(x) = \prod_{h \in \mathrm{ne}(x)\backslash\{f\}} \mu_{h\rightarrow x}(x)$$

factor $f \rightarrow$ **variable** x:

$$\mu_{f\rightarrow x}(x) = \sum_{\mathbf{x}\backslash x} \left(f(\mathbf{x}) \prod_{y \in \mathrm{ne}(f)\backslash\{x\}} \mu_{y\rightarrow f}(y) \right)$$

Propagation in Factor Graphs

We can compute probabilities in a factor graph by propagating messages from variable nodes to function nodes and *vice versa*.

$$\mathrm{ne}(x_i) \overset{\mathrm{def}}{=} \{f_j : x \in C_j\} \qquad \mathrm{ne}(f_j) \overset{\mathrm{def}}{=} \{x_i : x_i \in C_j\}$$

variable $x \rightarrow$ **factor** f:

$$\mu_{x\rightarrow f}(x) = \prod_{h\in\mathrm{ne}(x)\setminus\{f\}} \mu_{h\rightarrow x}(x)$$

factor $f \rightarrow$ **variable** x:

$$\mu_{f\rightarrow x}(x) = \sum_{\mathbf{x}\setminus x} \left(f(\mathbf{x}) \prod_{y\in\mathrm{ne}(f)\setminus\{x\}} \mu_{y\rightarrow f}(y) \right)$$

If a variable has only one factor as a neighbor, it can initiate message propagation.

Propagation in Factor Graphs

We can compute probabilities in a factor graph by propagating messages from variable nodes to function nodes and *vice versa*.

$$\mathrm{ne}(x_i) \overset{\mathrm{def}}{=} \{f_j : x \in C_j\} \qquad \mathrm{ne}(f_j) \overset{\mathrm{def}}{=} \{x_i : x_i \in C_j\}$$

variable $x \longrightarrow$ **factor** f:

$$\mu_{x \to f}(x) = \prod_{h \in \mathrm{ne}(x) \backslash \{f\}} \mu_{h \to x}(x)$$

factor $f \longrightarrow$ **variable** x:

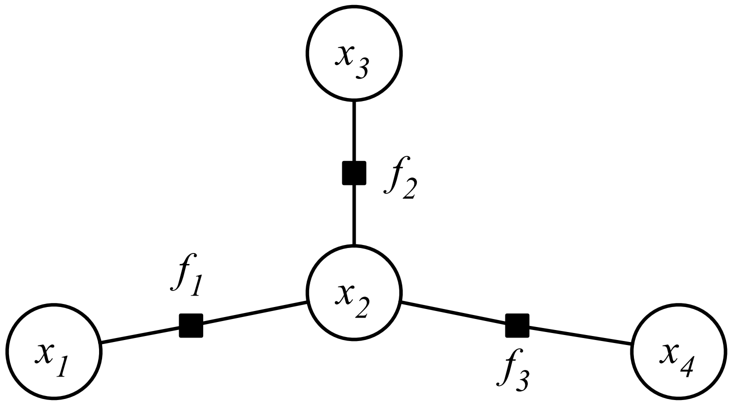
$$\mu_{f \to x}(x) = \sum_{\mathbf{x} \backslash x} \left(f(\mathbf{x}) \prod_{y \in \mathrm{ne}(f) \backslash \{x\}} \mu_{y \to f}(y) \right)$$

If a variable has only one factor as a neighbor, it can initiate message propagation.

Once a variable has received all messages from its neighboring function nodes we can compute the probability of that variable by multiplying all the messages and renormalising:

$$p(x) \propto \prod_{h \in \mathrm{ne}(x)} \mu_{h \to x}(x)$$

Propagation in Factor Graphs



initialise all messages to be 1

an example schedule of messages resulting in computing $p(x_4)$:

message direction	message value
$x_1 \to f_1$	$1(x_1)$
$x_3 \to f_2$	$1(x_3)$
$f_1 \to x_2$	$\sum_{x_1} f_1(x_1, x_2) 1(x_1)$
$f_2 \to x_2$	$\sum_{x_3} f_2(x_3, x_2) 1(x_3)$
$x_2 \to f_3$	$\left(\sum_{x_1} f_1(x_1, x_2)\right) \left(\sum_{x_3} f_2(x_3, x_2)\right)$
$f_3 \to x_4$	$\sum_{x_2} f_3(x_2, x_4) \left(\sum_{x_1} f_1(x_1, x_2)\right) \left(\sum_{x_3} f_2(x_3, x_2)\right)$

Elimination Rules for Factor Graphs

- **eliminating observed variables**

  If a variable x_i is **observed**, i.e. its value is given, then it is a *constant* in all functions that include x_i.

  We can **eliminate** x_i from the graph by removing the corresponding node and modifying all neighboring functions to treat it as a constant.

Elimination Rules for Factor Graphs

- **eliminating hidden variables**

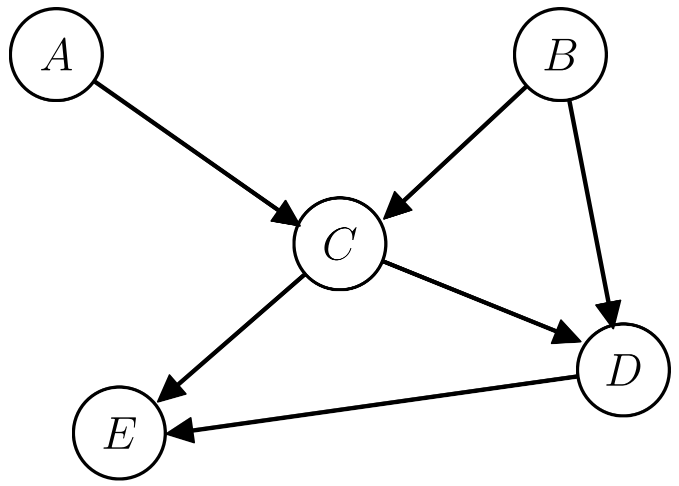
  If a variable x_i is **hidden** and we are not interested in it we can eliminate it from the graph by summing over all its values.

$$\begin{aligned}\sum_{x_i} p(\mathbf{x}) &= \frac{1}{Z}\sum_{x_i}\prod_j f_j(\mathbf{x}_{C_j}) \\ &= \frac{1}{Z}\prod_{j\notin \mathrm{n}(x_i)} f_j(\mathbf{x}_{C_j}) \left(\sum_{x_i}\prod_{k\in \mathrm{n}(x_i)} f_k(\mathbf{x}_{C_k})\right) \\ &= \frac{1}{Z}\prod_{j\notin \mathrm{n}(x_i)} f_j(\mathbf{x}_{C_j}) \quad f_{\mathrm{new}}(\mathbf{x}_{C_{\mathrm{new}}})\end{aligned}$$

where $f_{\mathrm{new}}(\mathbf{x}_{C_{\mathrm{new}}}) = \sum_{x_i}\prod_{k\in \mathrm{n}(x_i)} f_k(\mathbf{x}_{C_k})$ and $C_{\mathrm{new}} = \bigcup_{k\in \mathrm{n}(x_i)} C_k \setminus \{i\}$.

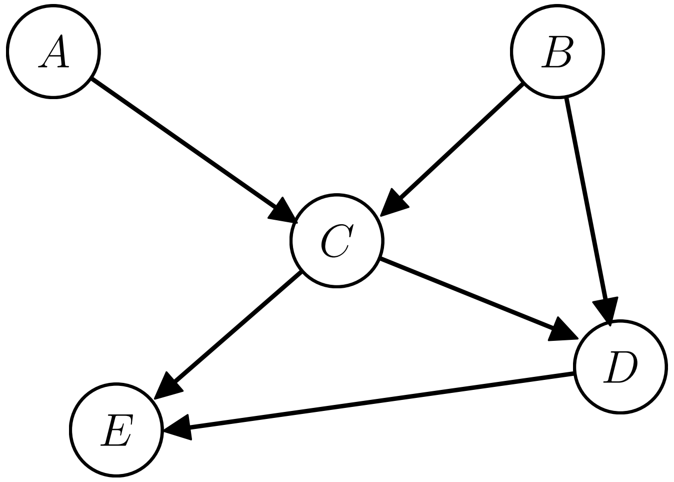
This causes all its neighboring function nodes to merge into one new function node.

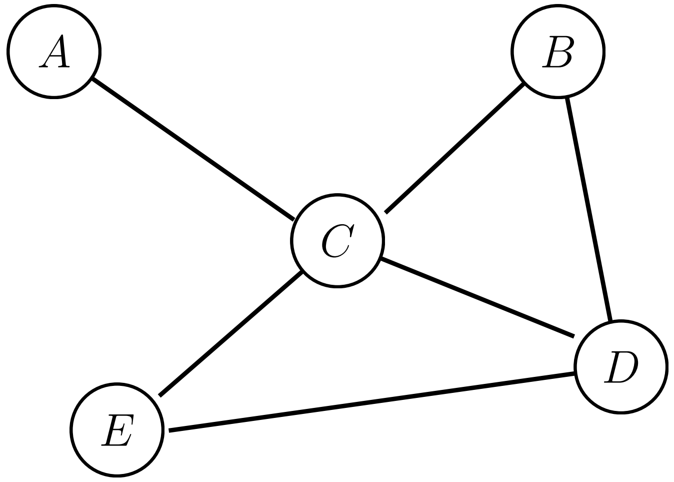
Case IV: DAGs



To reduce a multiply-connected DAG to a tree, we need to convert to the equivalent undirected graph. Again, we give up some knowledge of conditional independence structure in exchange for tractability.

Case IV: DAGs

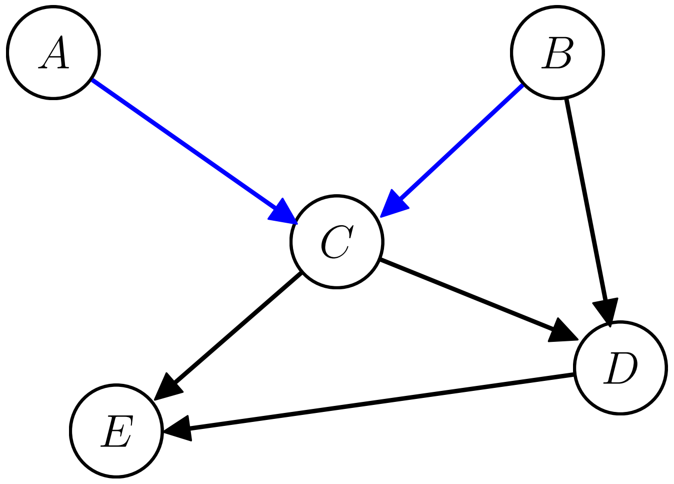


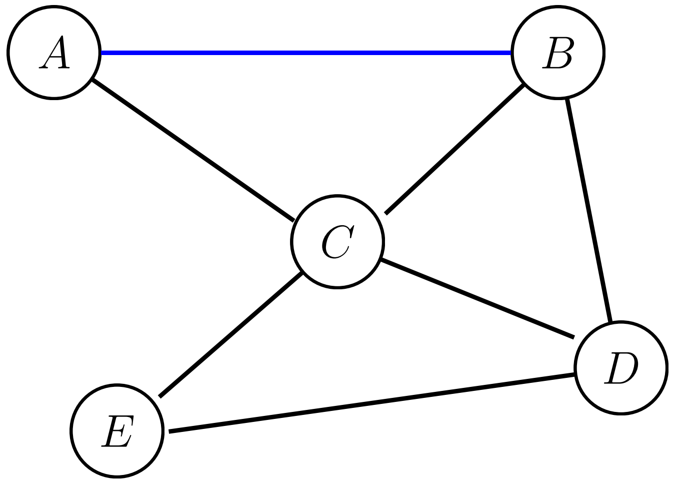


To reduce a multiply-connected DAG to a tree, we need to convert to the equivalent undirected graph. Again, we give up some knowledge of conditional independence structure in exchange for tractability.

To do this, replace each directed edge by an undirected edge, and . . .

Case IV: DAGs



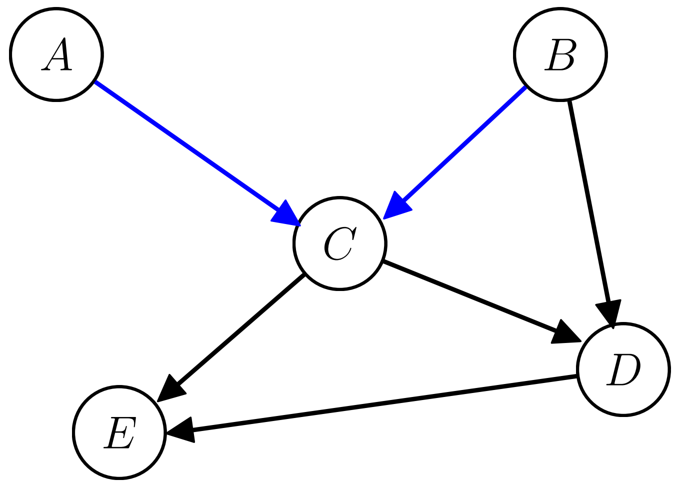


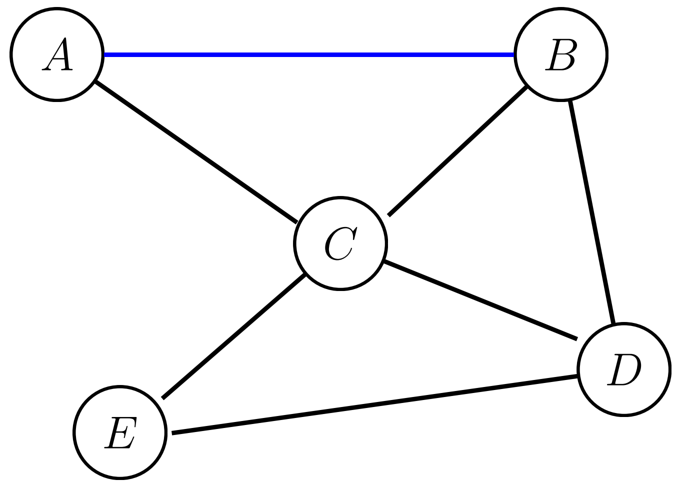
To reduce a multiply-connected DAG to a tree, we need to convert to the equivalent undirected graph. Again, we give up some knowledge of conditional independence structure in exchange for tractability.

To do this, replace each directed edge by an undirected edge, and . . .

“Moralise” by marrying unconnected co-parents.

Case IV: DAGs





To reduce a multiply-connected DAG to a tree, we need to convert to the equivalent undirected graph. Again, we give up some knowledge of conditional independence structure in exchange for tractability.

To do this, replace each directed edge by an undirected edge, and . . .

"Moralise" by marrying unconnected co-parents.

The resulting undirected graph may then need to be triangulated to form the junction tree.

Other approaches for multiply connected Bayes Nets

In some cases, the knowledge of conditional independence thrown away in the Junction Tree algorithm makes the problem too difficult. Cliques in the eventual tree become too large (indeed, the eventual Markov Net might be full connected). Fortunately, other schemes are available:

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 singly connected. For each value of these variables run belief propagation on the singly connected network. Average the resulting beliefs with the appropriate weights.

Loopy Belief Propagation: just use BP although there are loops. In this case the terms “upstream” and “downstream” are not clearly defined. 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.