Probabilistic & Unsupervised Learning

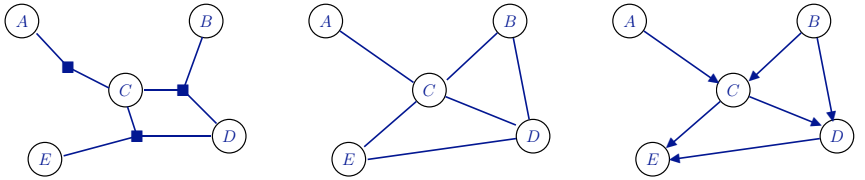
Graphical Models

Yee Whye Teh
ywteh@gatsby.ucl.ac.uk

Gatsby Computational Neuroscience Unit
University College London

Term 1, Autumn 2008

Three Kinds of Graphical Models



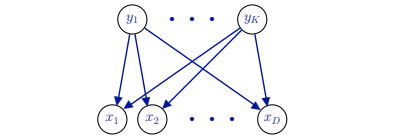
factor graph | undirected graph | directed graph

Nodes in graph correspond to random variables.

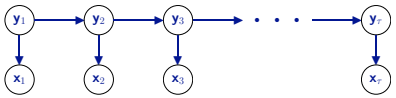
Edges in graph correspond to statistical dependencies between the corresponding variables.

(Absence of edges correspond to conditional independencies between variables).

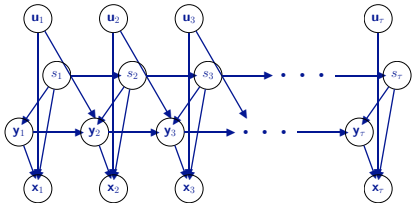
Some Examples



factor analysis
probabilistic PCA



hidden Markov models
linear dynamical systems



switching state-space models

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)
- Graphical models are a precise language to describe these relationships between variables.
- 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-purpose 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

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

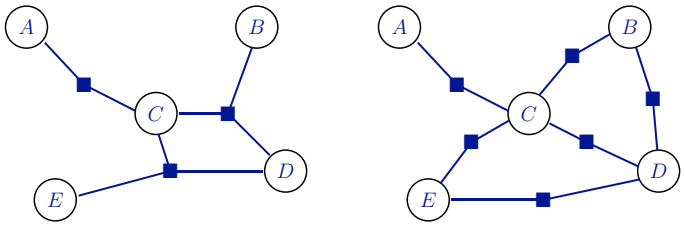
In general we can think of conditional independence between **sets of variables**:

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

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 squares represent factors in the joint distribution.

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

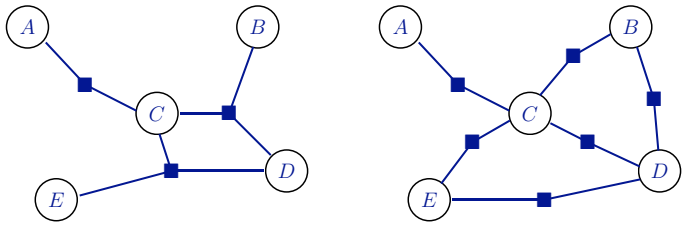
(b) $P(A,B,C,D,E) = \frac{1}{Z} f_1(A,C) f_2(B,C) f_3(C,D) f_4(B,D) f_5(C,E) f_6(D,E)$

The f_j are non-negative functions of their arguments, and Z is a normalization constant, e.g. in (a):

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

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

Factor Graphs



(a) (b)

In a **factor graph**, the probability distribution over variables in a factor graph factorizes:

$$p(\mathbf{X}) = \frac{1}{Z} \prod_j f_j(\mathbf{X}_{C_j})$$

where $\mathbf{X} = (X_1, \ldots, X_K)$, $\mathbf{X}_S = (X_i : i \in S)$, j indexes the factors f_j, C_j are the indices of variables adjacent to factor j, and Z is a normalization constant.
Factors describe compatibility between adjacent variables.

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: $X \perp\!\!\!\perp Y|\,\mathrm{ne}(X), \ \ \forall Y \notin \{X \cup \mathrm{ne}(X)\}$.

Proving Conditional Independence

Assume the following factorization:

$$P(X,Y,V) = \frac{1}{Z} g_1(X,V) g_2(Y,V)$$

Let's show that the above implies the conditional independence:

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

Summing over X,

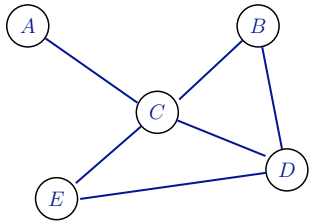
$$p(Y,V) = \frac{1}{Z} \left(\sum_X g_1(X,V) \right) g_2(Y,V)$$

So dividing $P(X,Y,V)$ by $P(Y,V)$ gives

$$p(X|Y,V) = \frac{p(X,Y,V)}{p(Y,V)} = \frac{g_1(X,V)}{\sum_{X'} g_1(X',V)}$$

Since the RHS does not depend on Y, it follows that X is independent from Y given V.

Undirected Graphical Models



In an **undirected graphical model**, the joint probability over all variables can be written in a factored form:

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

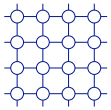
where C_j are the **maximal cliques** of the undirected graph.
(Cliques are fully connected subgraphs, maximal cliques are cliques not contained in other cliques).

Undirected graphical models are also called **Markov networks**.

Undirected graphical models are very similar to factor graphs (not quite equivalent).

Examples of Undirected Graphical Models

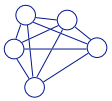
- Markov random fields (used in computer vision)



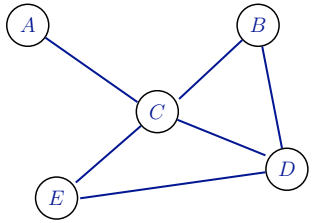
- Maximum entropy language models (used in speech and language modelling)

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

- Conditional random fields are undirected graphical models (conditioned on the input variables).
- Boltzmann machines (a kind of neural network/Ising model)



Undirected Graphical Models



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

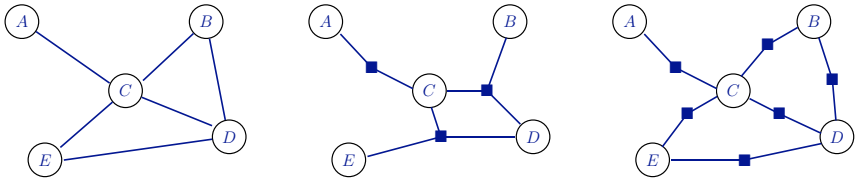
Fact: $X \perp\!\!\!\perp Y|\mathcal{V}$ if every path between X and Y contains some node $V \in \mathcal{V}$

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

$\mathcal{V}$ 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.

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.

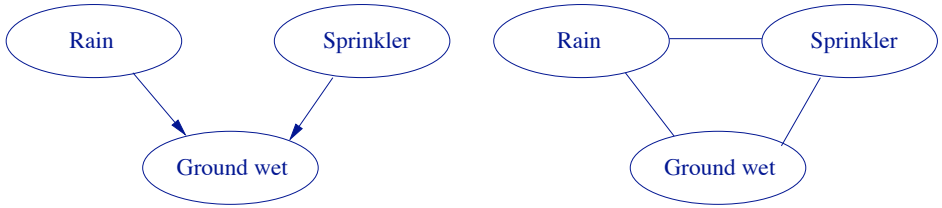
(c) also represents the fact that the probability factors into a product of pairwise functions.

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 $\mathcal{O}(K^2)$.

Factor graphs have richer expressive power than undirected graphical models.

Problems with Undirected Graphs and Factor Graphs

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

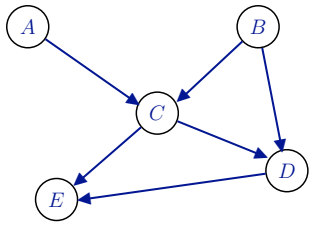


This highlights the difference between **marginal independence** and **conditional independence**.

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



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

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

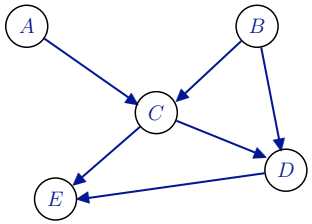
In general:

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

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

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

Conditional Independence in Directed Acyclic Graphs

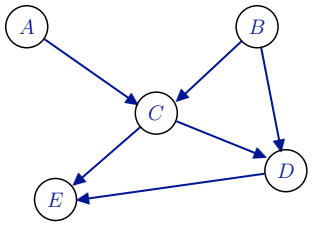


Reading conditional independence from DAGs is more complicated than in undirected graphs.

- $A \perp\!\!\!\perp E \mid \{B, C\}$: 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. Say that the path is **blocked** if there is a node W on the path such that either:

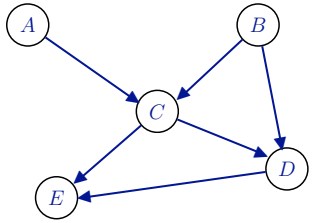
- W has convergent arrows ($\rightarrow W \leftarrow$) on the path and **neither** W **nor its descendents are in** $\mathcal{V}$. W is also called a collider node.
- or W does not have convergent arrows ($\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow$) and $W \in \mathcal{V}$. This is similar 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: $\{\text{parents}(X) \cup \text{children}(X) \cup \text{parents-of-children}(X)\}$.

[1] A path in the DAG ignoring the direction of edges.

The Bayes-ball algorithm



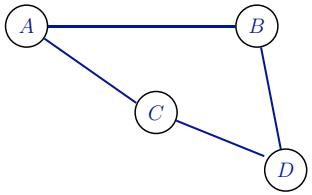
Game: can you get a ball from X to Y without being blocked by $\mathcal{V}$? If so, $X \not\perp\!\!\!\perp Y|\mathcal{V}$.

Rules: the ball can only be passed along or bounced back under the following conditions:

- If $W \notin \mathcal{V}$ and either $\rightarrow W \rightarrow$ or $\leftarrow W \leftarrow$, then the ball is passed along.
- If $W \notin \mathcal{V}$, then the ball is bounced from any child to any child (including the child it came from).
- If $W \in \mathcal{V}$, then the ball is bounced from any parent to any parent (including the parent it came from).

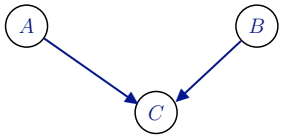
Note: if $W \in \mathcal{V}$ it blocks all balls from its children, and balls from parents are blocked from reaching its children.

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 $\Longrightarrow$ dependence in DAG but independence in undirected graph.



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

Graphs, Conditional Independencies, and Families of Distributions

Corresponding to each (factor, undirected, directed acyclic) graph G is a set of conditional independency statements $\mathcal{C}_G = \{X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i\}$.

Corresponding to G is also a parameterized family of distributions $\mathcal{P}_G$, e.g. for factor graph

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

Corresponding to any set of conditional independency statements $\mathcal{C}$ is also a family of distributions satisfying all statements in $\mathcal{C}$:

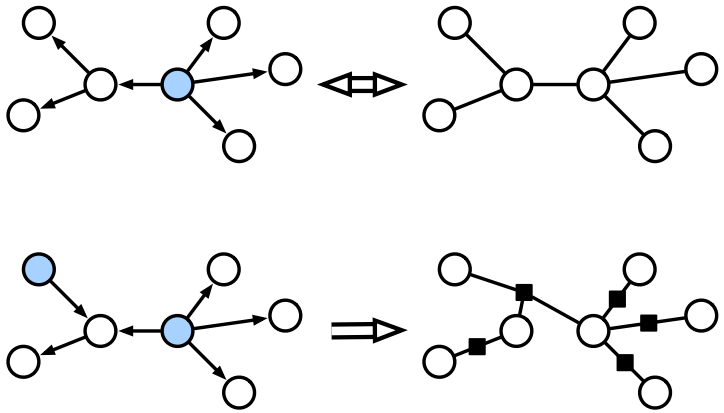
$$\mathcal{P}_{\mathcal{C}} = \{p(\mathbf{X}) \ : \ p(X_i, Y_i|\mathcal{V}_i) = p(X_i|\mathcal{V}_i)p(Y_i|\mathcal{V}_i) \text{ for all } X_i \perp\!\!\!\perp Y_i|\mathcal{V}_i \text{ in } \mathcal{C} \ \}$$

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

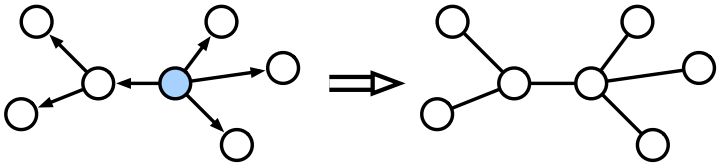
Summary

- Three kinds of graphical models: directed acyclic, undirected, factor graphs. There are other important classes, e.g. directed mixed graphs.
- Conditional independencies and marginal independencies.
- Markov blanket, boundaries and d-separation.
- Differences between all three types of graphical models.
- Next up: tree-structured graphical models, message-passing algorithms.

Tree-structured Graphical Models

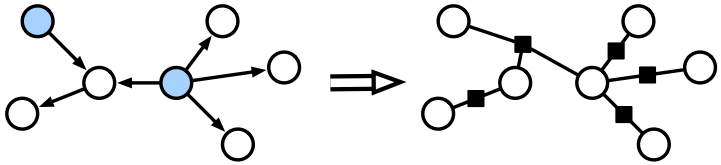


Directed Rooted Trees to Undirected Trees



$$p(\mathbf{X}) = p(X_r) \prod_{i \neq r} p(X_i | X_{\text{pa}(i)})$$
$$= \prod_{\text{edges } (ij)} f_{(ij)}(X_i, X_j)$$

Polytrees to Tree-structured Factor Graphs



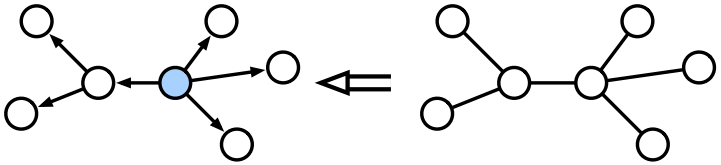
Polytrees are tree-structured DAGs with potentially multiple roots.

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

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 to Directed Rooted Trees



This direction is trickier; if we can compute the marginal distributions on single nodes $p(X_i)$ and on edges $p(X_i, X_j)$, then the DAG is parameterized as:

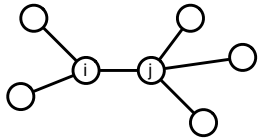
$$p(\mathbf{X}) = p(X_r) \prod_{i \neq r} p(X_i | X_{\text{pa}(i)})$$
$$= p(X_r) \prod_{i \neq r} \frac{p(X_i, X_{\text{pa}(i)})}{p(X_{\text{pa}(i)})}$$
$$= \frac{\prod_{\text{edges } (ij)} p(X_i, X_j)}{\prod_{\text{nodes } i} p(X_i)^{\text{deg(i)-1}}}$$

Picking any node as root and pointing all edges away from it (computing conditional distributions using above) gives the DAG parameterization.
How do we compute $p(X_i)$ and $p(X_i, X_j)$? **Belief propagation**!

Belief Propagation on Undirected Trees

Undirected tree parameterization for joint distribution:

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



Define $T_{i \to j}$ to be the subtree containing i if j is removed. Define **messages**:

$$M_{i \to j}(X_j) = \sum_{X_{T_{i \to j}}} f_{(ij)}(X_i, X_j) \prod_{\text{edges } (i'j') \text{ in } T_{i \to j}} f_{(i'j')}(X_{i'}, X_{j'})$$

Then the messages can be recursively computed as follows:

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

and:

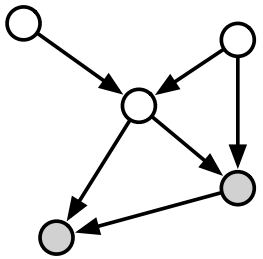
$$p(X_i) \propto \prod_{k \in \text{ne}(i)} M_{k \to i}(X_i)$$

$$p(X_i, X_j) \propto f_{(ij)}(X_i, X_j) \prod_{k \in \text{ne}(i) \setminus j} M_{k \to i}(X_i) \prod_{l \in \text{ne}(j) \setminus i} M_{l \to j}(X_j)$$

Inference in Graphical Models

Thus far, we have used graphical models to encode the conditional independencies and parameterizations 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.



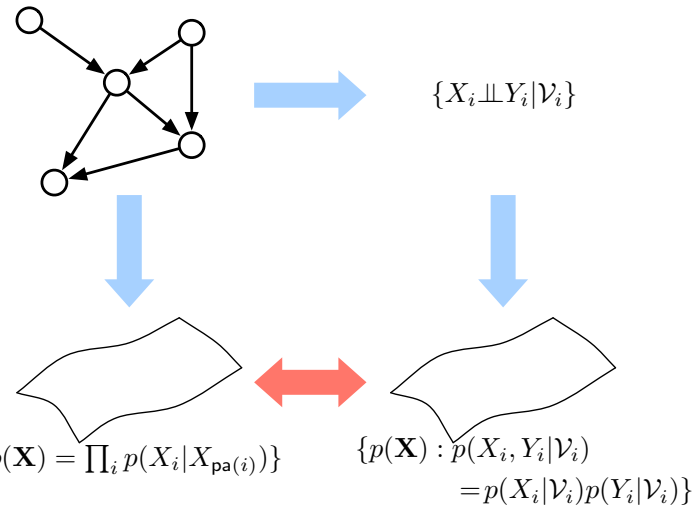
- marginals: $p(A|DE), \ldots$.
- most likely values: argmax $p(ABC|DE)$

Message passing algorithms exploit conditional independence relationships to make this computation efficient. Examples:

- forward-backward on HMMs and SSMs,
- Viterbi on HMMs and SSMs,
- Belief Propagation on undirected trees.

Today we will learn about message-passing algorithms that can work on arbitrary graphs. Specifically we will try to compute marginal distributions over single variables.

Recap on Graphical Models



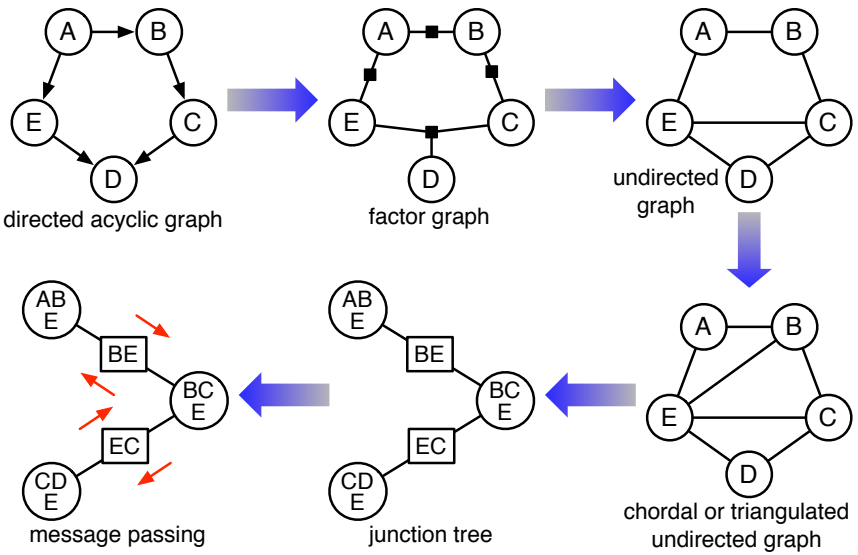
Graph Transformations

Our general approach to inference in arbitrary graphical models is to transform these graphical models to ones belonging to an easy-to-handle class (specifically, **junction or join trees**).

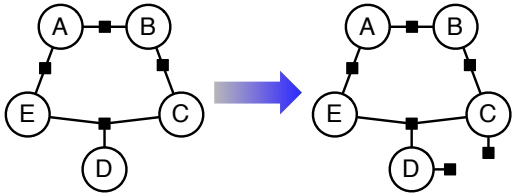
Suppose we are interested in inference in a distribution $p(\mathbf{X})$ representable in a graphical model G.

- In transforming G to an easy-to-handle G', we need to ensure that $p(\mathbf{X})$ is representable in G' too.
- This can be ensured by making sure that every step of the graph transformation only removes conditional independencies, never adds them.
- This guarantees that the family of distributions can only grow at each step, and $p(\mathbf{X})$ will be in the family of distributions represented by G'.
- Thus inference algorithms working on G' will work for $p(\mathbf{X})$ too.

The Junction Tree Algorithm



Entering Evidence in Factor Graphs



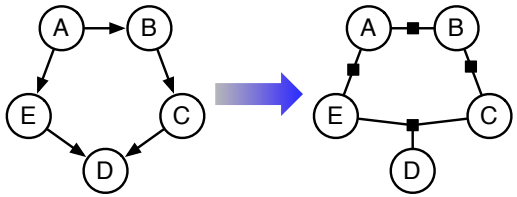
Often times we are interested in inferring posterior distributions given observed evidence (e.g. $D =$ wet and $C =$ rain).

This can be achieved by adding factors with just one adjacent node, with

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

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

Directed Acyclic Graphs to Factor Graphs



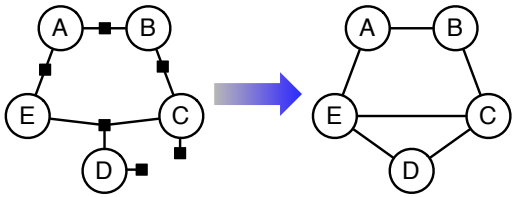
Factors are simply the conditional distributions in the DAG.

$$\begin{aligned} p(\mathbf{X}) &= \prod_i p(X_i | X_{\text{pa}(i)}) \\ &= \prod_i f_i(X_{C_i}) \end{aligned}$$

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.

Factor Graphs to Undirected Graphs



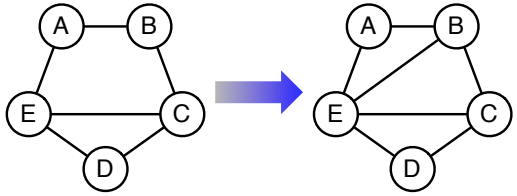
Just need to make sure that every factor is contained in some maximal clique of the undirected graph.

$$p(\mathbf{X}) = \frac{1}{Z} \prod_i f_i(X_{C_i})$$

We can make sure of this simply by converting each factor into a clique, and absorbing $f_i(X_{C_i})$ into the factor of some maximal clique containing it.

The transformation DAG $\Rightarrow$ undirected graph is called **moralization**—we simply "marry" all parents of each node by adding edges connecting them, then drop all arrows on edges.

Triangulation of Undirected Graphs



Message passing—messages contain information from other parts of the graph, and this information is propagated around the graph during inference.

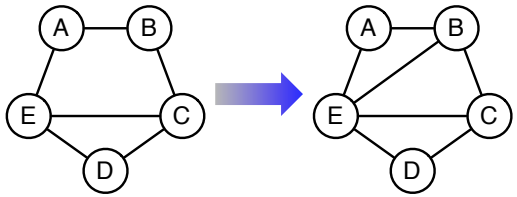
If loops (cycles) are present in the graph, message passing can lead to overconfidence due to double counting of information, and to oscillatory (non-convergent) behaviour[2].

To prevent this overconfident and oscillatory behaviour, we need to make sure that different channels of information communicate with each other to prevent double counting.

Triangulation: add edges to the graph so that every loop of size > 4 has at least one chord. Note recursive nature: adding edges often creates new loops; we need to make sure new loops of length > 4 have chords too.

[2]This is called **loopy belief propagation**, and we will see in the second half of the course that this is an important class of approximate inference algorithms.

Triangulation of Undirected Graphs



Triangulation: add edges to the graph so that every loop of size > 4 has at least one chord.

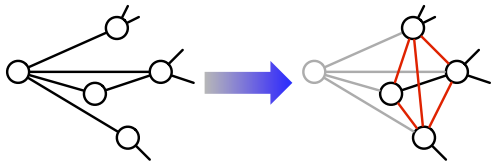
Remember that adding edges always removes conditional independencies and enlarges the family of distributions.

There are many ways to add chords; in general finding the best triangulation is NP-complete.

Here we describe one method of triangulation called **variable elimination**.

An undirected graph where every loop of size > 4 has at least one chord is called **chordal** or **triangulated**.

Variable Elimination of Undirected Graphs

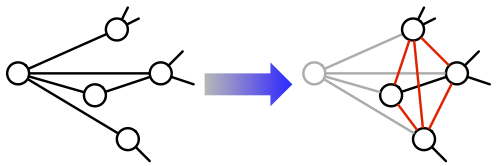


Say we compute the marginal distribution of a variable by brute force—sum out all other variables one by one (eliminate it from the graph).
Let the order of elimination be $X_{\sigma(1)}, X_{\sigma(2)}, \ldots, X_{\sigma(n)}$ with $X_{\sigma(n)}$ being the variable whose marginal distribution we are interested in.

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

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

Variable Elimination of Undirected Graphs



After we have eliminated all variables, go back to original graph, and add in all edges added during elimination.

Theorem: the graph with elimination edges added in is chordal.
Proof: by induction on the number of nodes in the graph.

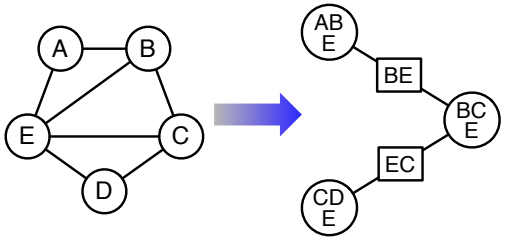
Finding a good triangulation is related to finding a good elimination ordering $\sigma(1), \ldots, \sigma(n)$. This is NP-complete.

Heuristics for good elimination ordering exist. Pick next variable to eliminate by:

- Minimum deficiency search: choose variable with least number of edges added.
- Maximum cardinality search: choose variable with maximum number of neighbours.

Minimum deficiency search has been empirically found to be better.

Chordal Graphs to Junction Trees



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

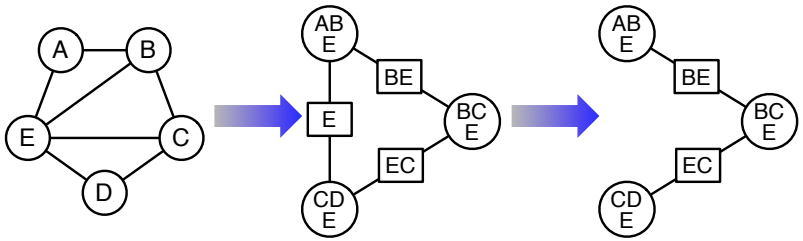
Variable sets on nodes are called cliques, and variable sets on edges are separators.

A junction tree has two properties:

- 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 Graphs to Junction Trees

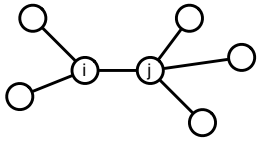


The following procedure converts a chordal graph to a junction tree:

1. Find the set of maximal cliques $C_1, \ldots, C_k$ (each of these cliques consists of an eliminated variable and its neighbours, so finding maximal cliques is easy).
2. Construct a weighted graph, with nodes labelled by the maximal cliques, and edges labelled by intersection of adjacent cliques.
3. Define the weight of an edge to be the size of the separator.
4. Run maximum weight spanning tree on the weighted graph.
5. The maximum weight spanning tree will be the junction tree.

Recap: Belief Propagation on Undirected Trees

Undirected tree parameterization for joint distribution:



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

Define $T_{i\to j}$ to be the subtree containing i if j is removed. Define **messages**:

$$M_{i\to j}(X_j) = \sum_{X_{T_{i\to j}}} f_{(ij)}(X_i, X_j) \prod_{\text{edges } (i'j') \text{ in } T_{i\to j}} f_{(i'j')}(X_{i'}, X_{j'})$$

Then the messages can be recursively computed as follows:

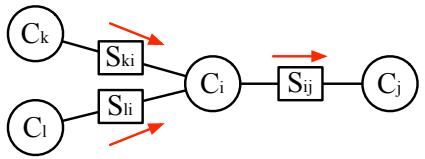
$$M_{i\to j}(X_j) = \sum_{X_i} f_{(ij)}(X_i, X_j) \prod_{k\in \mathsf{ne}(i)\setminus j} M_{k\to i}(X_i)$$

and:

$$p(X_i) \propto \prod_{k\in \mathsf{ne}(i)} M_{k\to i}(X_i)$$

$$p(X_i, X_j) \propto f_{(ij)}(X_i, X_j) \prod_{k\in \mathsf{ne}(i)\setminus j} M_{k\to i}(X_i) \prod_{l\in \mathsf{ne}(j)\setminus i} M_{l\to j}(X_j)$$

Message Passing on Junction Trees



Since maximal cliques in the chordal graph are nodes of the junction tree, we have:

$$p(\mathbf{X}) = \frac{1}{Z} \prod_i f_i(X_{C_i})$$

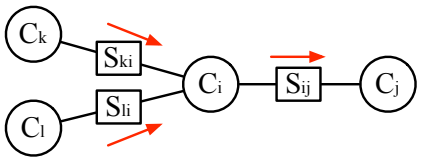
where C_i ranges over the cliques of the junction tree.

Let $S_{ij} = C_i \cap C_j$ be the separator between cliques i and j.
Let $T_{i\to j}$ be the union of cliques on the i side of j.

Define **messages**:

$$M_{i\to j}(X_{S_{ij}}) = \sum_{X_{T_{i\to j}\setminus C_j}} \prod_{k: C_k \subset T_{i\to j}} f_k(X_{C_k})$$

Message Passing on Junction Trees



Messages can be computed recursively by:

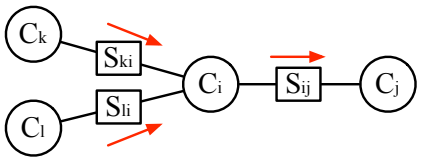
$$M_{i\to j}(X_{S_{ij}}) = \sum_{X_{C_i\setminus S_{ij}}} f_i(X_{C_i}) \prod_{k\in \mathsf{ne}(i)\setminus j} M_{k\to i}(X_{S_{ki}})$$

And marginal distributions on cliques and separators are:

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

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

Reparameterization on Junction Trees



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

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

Then our probability distribution is initially

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

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

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

Consistency and Parameterization on Junction Trees

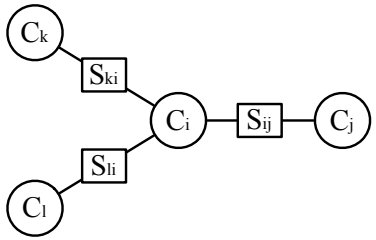
Because of the running intersection property and because junction trees are trees, **local consistency** of marginal distributions between cliques and separators guarantees **global consistency**.

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

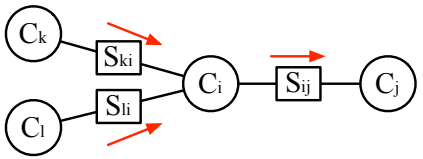
$$p(\mathbf{X}) = \frac{\prod_{\text{cliques } i} q_i(X_{C_i})}{\prod_{\text{separators } (ij)} r_{ij}(X_{S_{ij}})}$$



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

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

Reparameterization on Junction Trees



Some properties of Hugin propagation:

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

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

- 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 is $O(\sum_i |\mathcal{X}_{C_i}|)$. 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 eventhough 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.

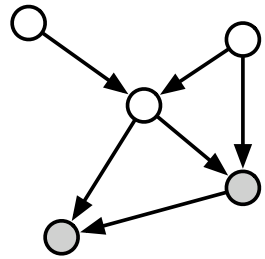
Second half of course: we will learn about a variety of approximate inference algorithms when the graphical model is so large/complex that no exact inference algorithm can work efficiently.

Learning in Graphical Models

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:

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



by EM, need to maximise

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

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