Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D-BSSE
Department of Biosystems
Science and Engineering

Exact inference in graphical models

Niko Beerenwinkel

Outline

- Exact inference in acyclic graphical models
  - Factor graphs
  - Message passing
  - Sum-product algorithm
- Exact inference in general models
  - Cluster trees, potentials
  - Message passing
  - Junction tree algorithm

Exact inference in *acyclic* graphical models

Markov random fields (MRFs)

- A MRF is an undirected graphical model, defined by the factorization over maximal cliques *C*

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C)$$

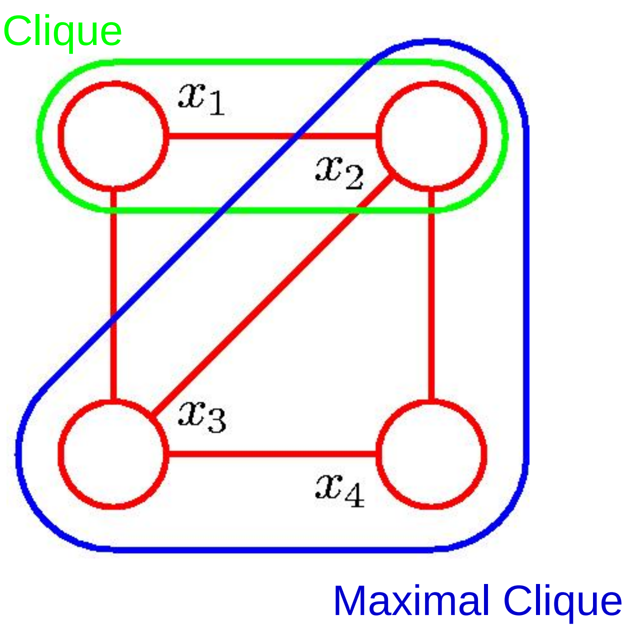
into clique potentials

$$\psi_C(\mathbf{x}_C) = \exp\{-E(\mathbf{x}_C)\}$$

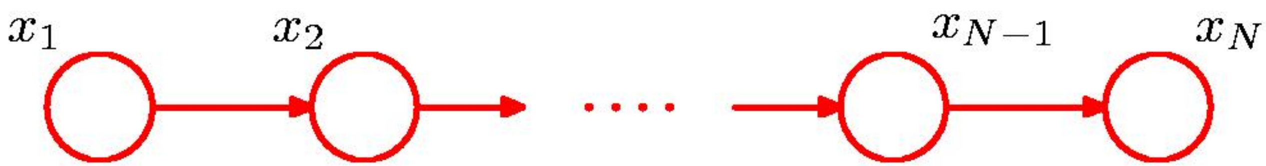
with the normalization coefficient

$$Z = \sum_{\mathbf{x}} \prod_C \psi_C(\mathbf{x}_C)$$

(Boltzmann distribution with energy *E*)



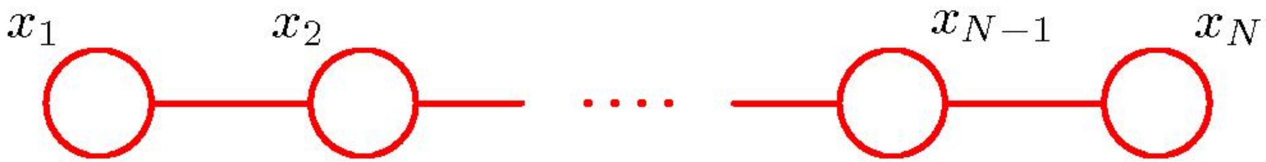
Directed vs undirected chains



$$p(\mathbf{x}) = \underbrace{p(x_1)p(x_2|x_1)} \; p(x_3|x_2) \cdots p(x_N|x_{N-1})$$

$$p(\mathbf{x}) = \frac{1}{Z} \psi_{1,2}(x_1, x_2) \; \psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N)$$

Inference in an undirected chain

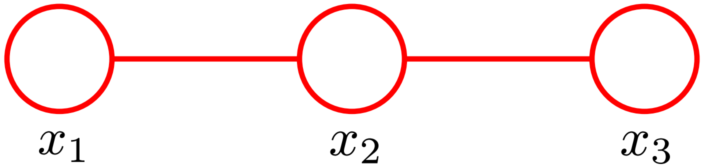


$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(\mathbf{x})$$

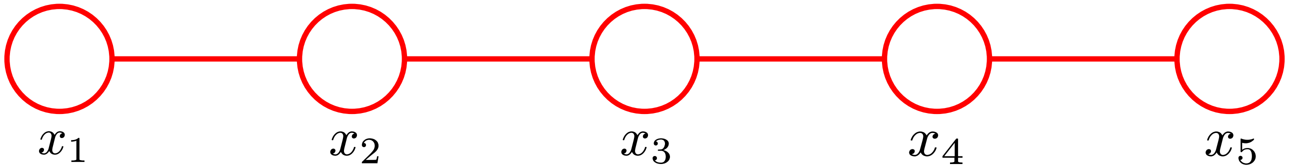
- Naïve implementation scales exponentially in chain length N.
- For directed chain models (Bayesian networks), we have computed this marginalization in time $O(K^2N)$ – forward algorithm!

Basic idea: distributive law, ab + ac = a(b + c)



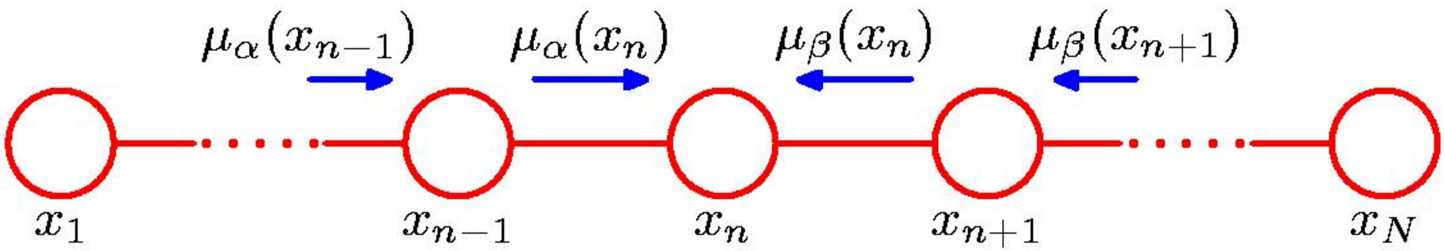
$$
\begin{aligned}
p(x_3) &= \sum_{x_1}\sum_{x_2} p(x_1, x_2, x_3) \\
&= \frac{1}{Z}\sum_{x_2}\sum_{x_1} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \\
&= \underbrace{\frac{1}{Z}\sum_{x_2} \psi_{2,3}(x_2, x_3) \underbrace{\sum_{x_1} \psi_{1,2}(x_1, x_2)}_{\mu_\alpha(x_2)}}_{\mu_\alpha(x_3)}
\end{aligned}
$$

Basic idea: distributive law, ab + ac = a(b + c)



$$\begin{aligned}
p(x_3) &= \frac{1}{Z} \sum_{x_1,x_2,x_4,x_5} \psi_{1,2}\psi_{2,3}\psi_{3,4}\psi_{4,5} \\
&= \frac{1}{Z} \sum_{x_1,x_2,x_4} \psi_{1,2}\psi_{2,3}\psi_{3,4} \sum_{x_5} \psi_{4,5} \\
&= \frac{1}{Z} \sum_{x_1,x_2} \psi_{1,2}\psi_{2,3} \sum_{x_4} \psi_{3,4} \sum_{x_5} \psi_{4,5} \\
&= \frac{1}{Z} \underbrace{\left[\sum_{x_2} \psi_{2,3} \sum_{x_1} \psi_{1,2} \right]}_{\mu_\alpha(x_3)} \underbrace{\left[\sum_{x_4} \psi_{3,4} \sum_{x_5} \psi_{4,5} \right]}_{\mu_\beta(x_3)}
\end{aligned}$$

Inference on a chain

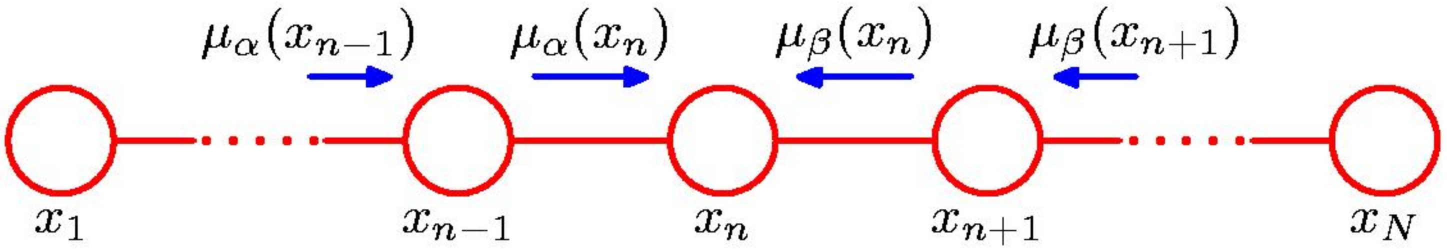


$$p(x_n) = \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_1} \psi_{1,2}(x_1, x_2) \right] \cdots \right]}_{\mu_\alpha(x_n)}$$

$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \cdots \right]}_{\mu_\beta(x_n)}$$

$O(NK^2)$

Message passing



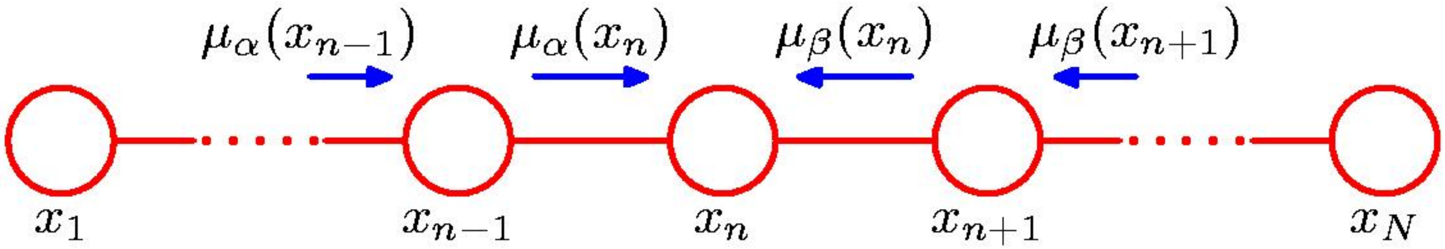
$$\begin{aligned}
\mu_\alpha(x_n) &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \left[\sum_{x_{n-2}} \cdots \right] \\
&= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1}).
\end{aligned}$$

$$\begin{aligned}
\mu_\beta(x_n) &= \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \left[\sum_{x_{n+2}} \cdots \right] \\
&= \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \mu_\beta(x_{n+1}).
\end{aligned}$$

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D-BSSE
Department of Biosystems
Science and Engineering

Initialization, termination



- Initialization

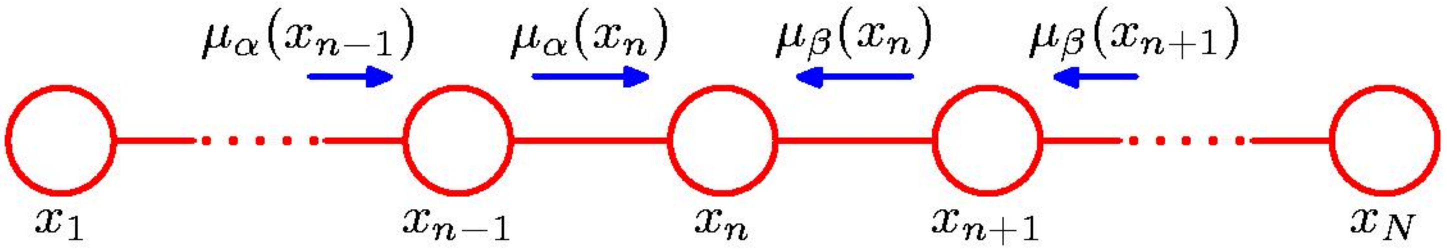
$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2) \qquad \mu_\beta(x_{N-1}) = \sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

- Normalization coefficient (partition function):

$$Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$$

- Each message consists of K values, multiplication is point-wise.

Summary (sum-product algorithm for chains)

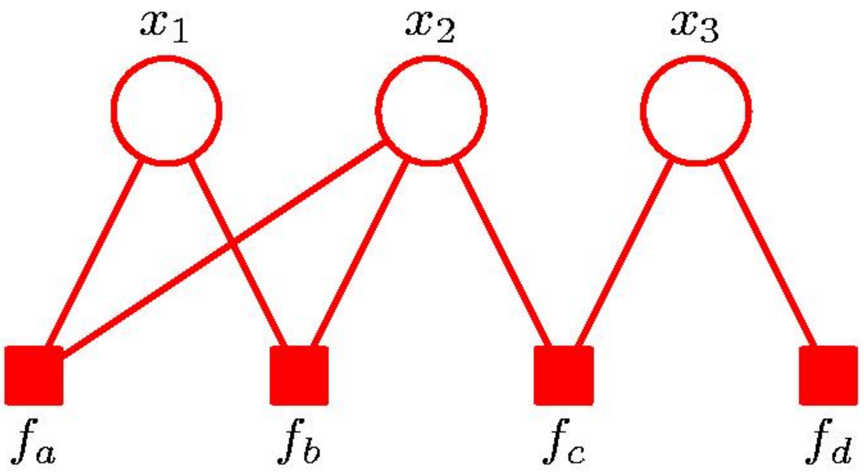


1. Compute and store all forward messages $\mu_\alpha(x_n)$
2. Compute and store all backward messages $\mu_\beta(x_n)$
3. Compute Z at any node x_m
4. Compute

$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

for all variables required.

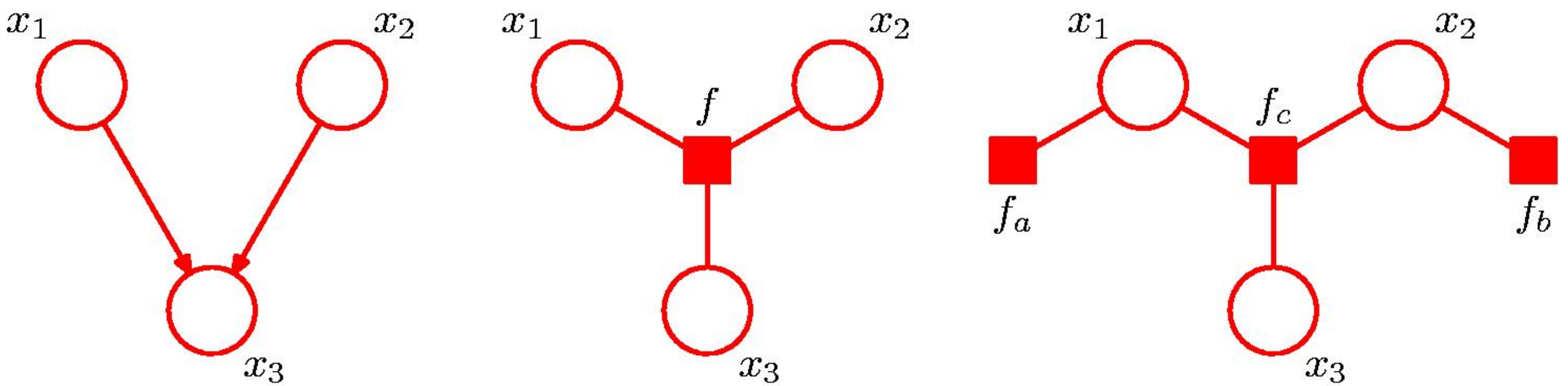
Factor graphs



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

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

Factor graphs from directed graphs

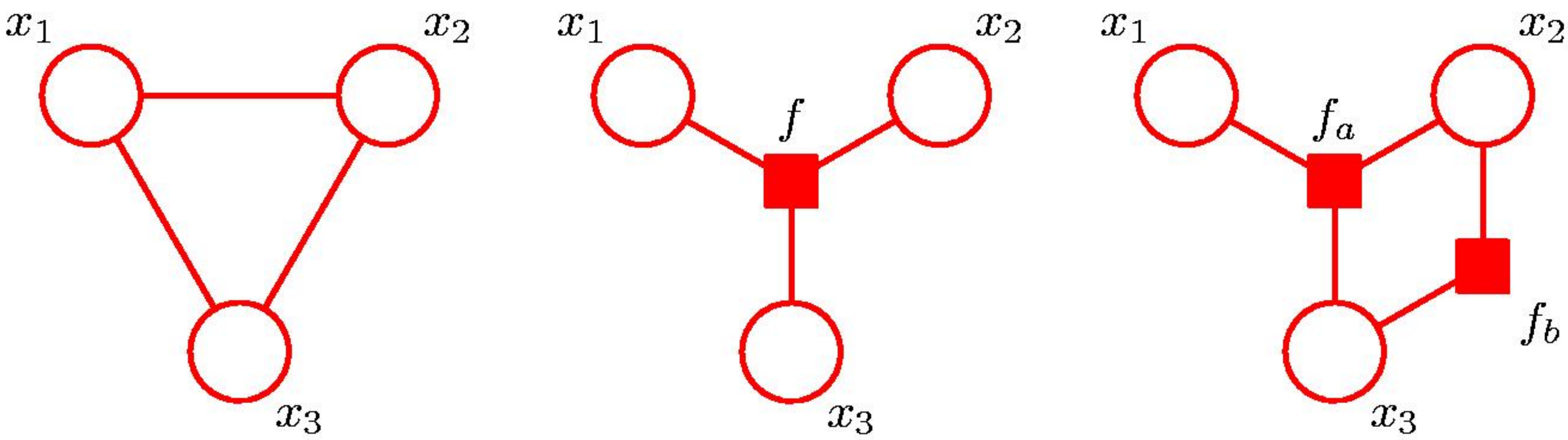


$$p(\mathbf{x}) = p(x_1)p(x_2)\,p(x_3|x_1,x_2)$$

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

$$\begin{aligned} f_a(x_1) &= p(x_1) \\ f_b(x_2) &= p(x_2) \\ f_c(x_1,x_2,x_3) &= p(x_3|x_1,x_2) \end{aligned}$$

Factor graphs from undirected graphs



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

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

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

Sum-product algorithm (belief propagation)

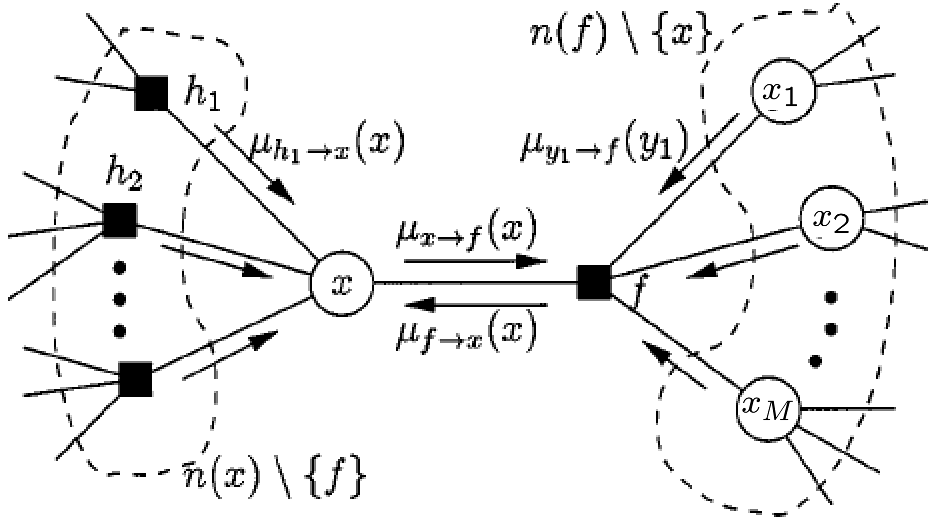
For *acyclic* factor graphs, to compute local marginals:

1. Pick an arbitrary node as root
2. Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
3. Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
4. Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D-BSSE
Department of Biosystems
Science and Engineering

Messages



- Variable to factor: $\mu_{x \to f}(x) := \prod_{h \in n(x) \setminus f} \mu_{h \to x}(x)$
- Factor to variable:

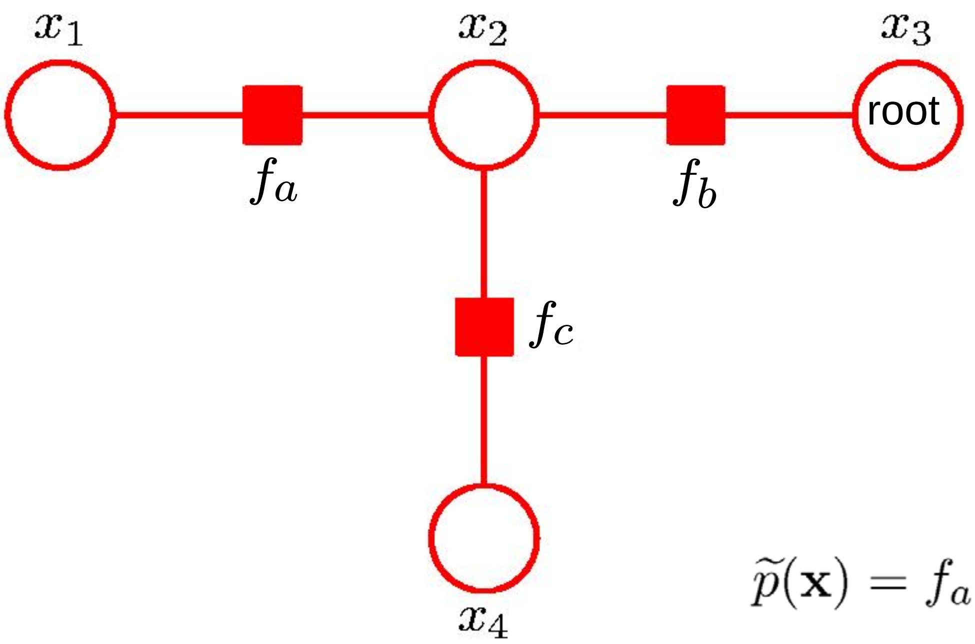
$$\mu_{f \to x}(x) := \sum_{x_1} \cdots \sum_{x_M} \left[f(x, x_1, \ldots, x_M) \prod_{x_m \in n(f) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

$$\mu_{f\to x}(x) = \sum_{x_1}\cdots\sum_{x_M}\left[f(x,x_1,\ldots,x_M)\prod_{x_m\in n(f)\setminus x}\mu_{x_m\to f}(x_m)\right]$$

Example (step 1)



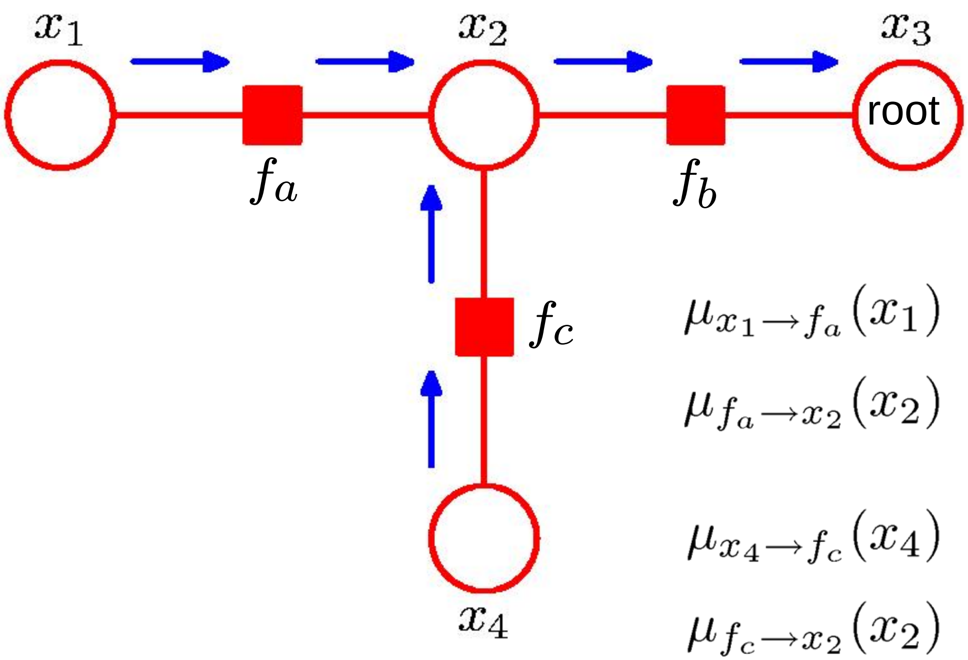
$$\widetilde{p}(\mathbf{x}) = f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_2,x_4)$$

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

$$\mu_{f \to x}(x) = \sum_{x_1} \cdots \sum_{x_M} \left[f(x, x_1, \ldots, x_M) \prod_{x_m \in n(f) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

Example (step 2)



$$\mu_{x_1 \to f_a}(x_1) = 1$$

$$\mu_{f_a \to x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2)$$

$$\mu_{x_4 \to f_c}(x_4) = 1$$

$$\mu_{f_c \to x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4)$$

$$\mu_{x_2 \to f_b}(x_2) = \mu_{f_a \to x_2}(x_2) \mu_{f_c \to x_2}(x_2)$$

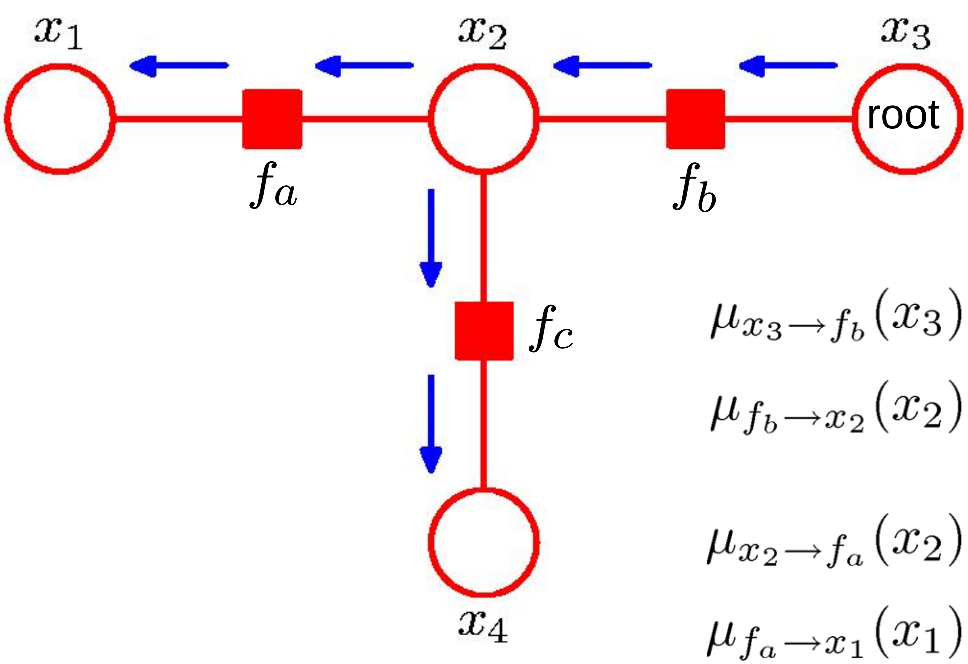
$$\mu_{f_b \to x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \to f_b}(x_2)$$

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

$$\mu_{f\to x}(x) = \sum_{x_1}\cdots\sum_{x_M}\left[f(x,x_1,\ldots,x_M)\prod_{x_m\in n(f)\setminus x}\mu_{x_m\to f}(x_m)\right]$$

Example (step 3)



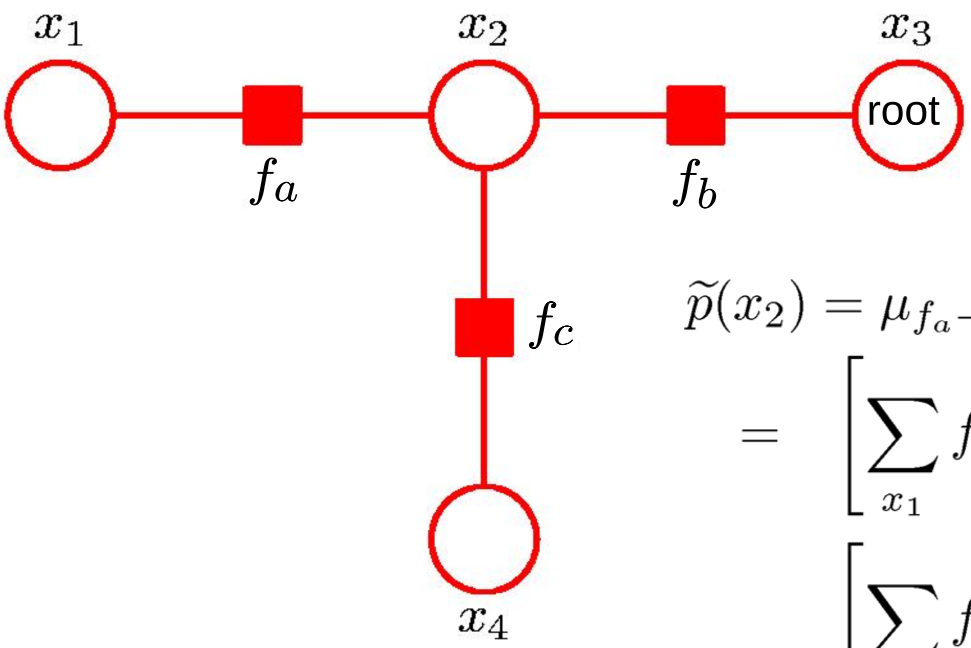
$$\begin{aligned}
\mu_{x_3\to f_b}(x_3) &= 1 \\
\mu_{f_b\to x_2}(x_2) &= \sum_{x_3} f_b(x_2,x_3) \\
\mu_{x_2\to f_a}(x_2) &= \mu_{f_b\to x_2}(x_2)\mu_{f_c\to x_2}(x_2) \\
\mu_{f_a\to x_1}(x_1) &= \sum_{x_2} f_a(x_1,x_2)\mu_{x_2\to f_a}(x_2) \\
\mu_{x_2\to f_c}(x_2) &= \mu_{f_a\to x_2}(x_2)\mu_{f_b\to x_2}(x_2) \\
\mu_{f_c\to x_4}(x_4) &= \sum_{x_2} f_c(x_2,x_4)\mu_{x_2\to f_c}(x_2)
\end{aligned}$$

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

$$\mu_{f\to x}(x) = \sum_{x_1}\cdots\sum_{x_M}\left[f(x,x_1,\ldots,x_M)\prod_{x_m\in n(f)\setminus x}\mu_{x_m\to f}(x_m)\right]$$

Example (step 4)



$$\begin{aligned}\widetilde{p}(x_2) &= \mu_{f_a\to x_2}(x_2)\mu_{f_b\to x_2}(x_2)\mu_{f_c\to x_2}(x_2)\\ &= \left[\sum_{x_1} f_a(x_1,x_2)\right]\left[\sum_{x_3} f_b(x_2,x_3)\right]\\ &\quad \left[\sum_{x_4} f_c(x_2,x_4)\right]\\ &= \sum_{x_1}\sum_{x_3}\sum_{x_4} f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_2,x_4)\\ &= \sum_{x_1}\sum_{x_3}\sum_{x_4}\widetilde{p}(\mathbf{x})\end{aligned}$$

Max-product algorithm

- To compute $\max_x p(x)$, the same factorization can be used.
- For chains,

$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} \ldots \max_{x_M} p(\mathbf{x})$$
$$= \frac{1}{Z} \max_{x_1} \cdots \max_{x_N} \left[\psi_{1,2}(x_1, x_2) \cdots \psi_{N-1,N}(x_{N-1}, x_N)\right]$$
$$= \frac{1}{Z} \max_{x_1} \left[\max_{x_2} \left[\psi_{1,2}(x_1, x_2) \left[\cdots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right] \cdots\right]\right]$$

- This is the Viterbi algorithm!

Max-product/sum algorithm

- Use the distributive laws

  a max(b, c) = max(ab, ac), a ≥ 0

  a + max(b, c) = max (a + b, a + c)

  For general loop-free factor graphs,

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_n} \prod_{f_s \in \mathrm{ne}(x_n)} \max_{X_s} f_s(x_n, X_s)$$

- sum-product ↔ max-product ↔ max-sum
  - same structure
  - even same implementation possible (using operator overloading)

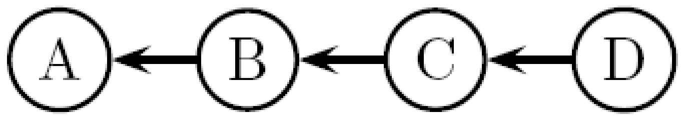
Exact inference in *general* graphical models

Exact inference in general DAGs

- Let U = {A, B, C, ...} be the "universe" of random variables A, B, C, ...
- Let P(U) be defined by a Bayesian network (G, θ).
- The junction tree algorithm is an efficient procedure for computing marginals of P(U).
- It is a direct extension of the sum-product algorithm to general DAGs.

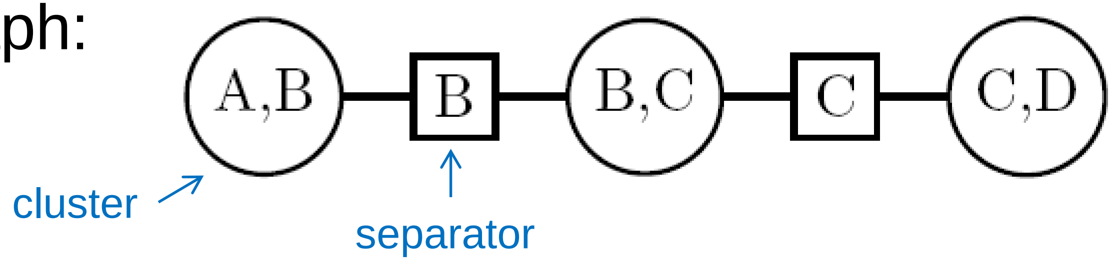
Cluster graph

- For U = {A, B, C, D}, consider the directed chain



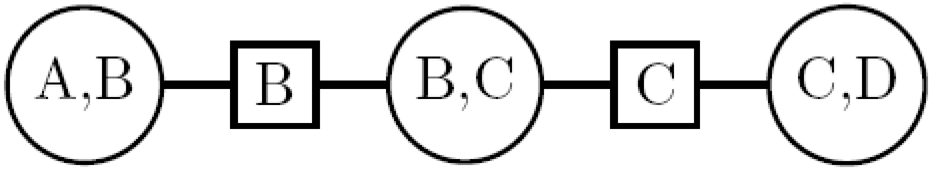
$$P(U) = P(A \mid B)P(B \mid C)P(C \mid D)P(D)$$

- Cluster graph:



with potentials ψ(A,B), ψ(B,C), ψ(C,D),
and separator potentials ψ(B), ψ(C).

Cluster potential representation



$$P(U) = \frac{\psi(A,B)\psi(B,C)\psi(C,D)}{\psi(B)\psi(C)}$$

$\longleftarrow$ cluster potentials

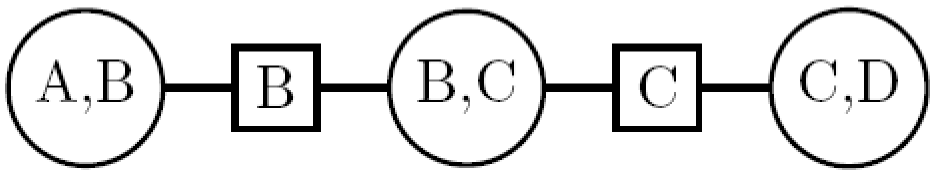
$\longleftarrow$ separator potentials

$$= P(A \mid B)P(B \mid C)P(C \mid D)P(D)$$

- This equality holds, for example, for ψ(A,B) = P(A | B), ψ(B,C) = P(B | C), ψ(C,D) = P(C,D), and ψ(B) = ψ(C) = 1.

Clique marginals

- For every singly-connected graph, P(U) can be represented as the product of the clique marginals divided by the product of the separator marginals.

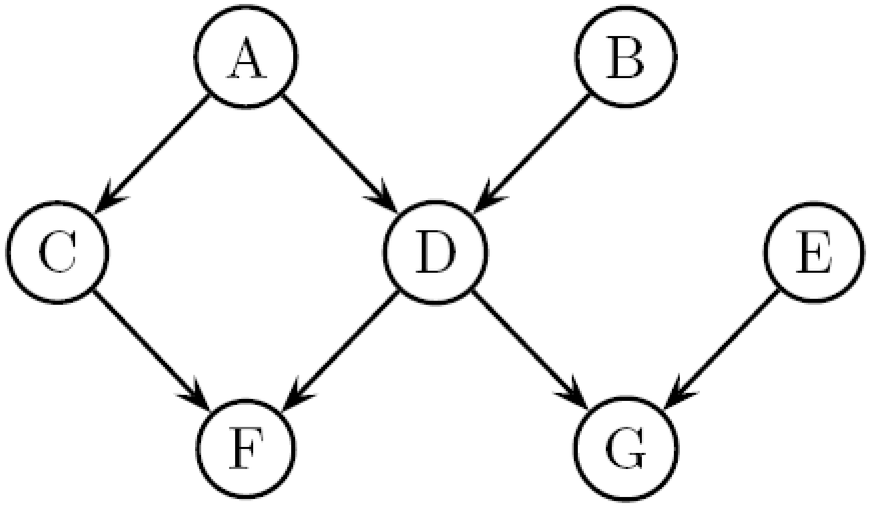


$$
\begin{aligned}
P(U) &= \frac{\psi^*(A,B)\psi^*(B,C)\psi^*(C,D)}{\psi^*(B)\psi^*(C)} \\
&= \frac{P(A,B)P(B,C)P(C,D)}{P(B)P(C)}
\end{aligned}
$$

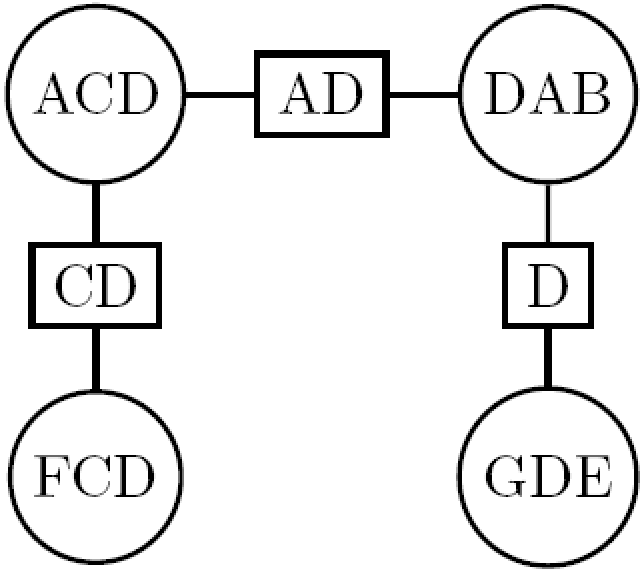
Cluster trees

- A *cluster* is a non-empty subset of U, represented by a node. For example, v = {A,C,D} $\subset$ U.
- A *separator* is the intersection of two adjacent nodes. For example, the separator of v = {D, A, B} and w = {A, C, D} is s = v $\cap$ w = {A, D}.
- A *cluster tree* is a tree whose nodes are clusters such that the union of all clusters is U.
- Associated with each node and with each separator is a potential ψ.
- We use the cluster tree to represent P(U).

Example



DAG



Cluster tree

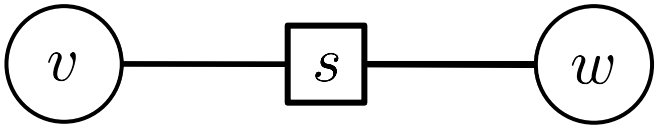
Cluster tree construction

1. Form a family of nodes V such that for each variable A $\in$ U, there is a node v $\in$ V with {A} $\cup$ pa(A) $\subset$ v.
2. Build a tree on V with separators.
3. Initialize all cluster and separator potentials to ψ = 1.
4. To each variable A, assign exactly one node v containing {A} $\cup$ pa(A), and update ψ(v) $\leftarrow$ ψ(v) P(A | pa(A)).

- Then

$$\prod_{v \in V} \psi(v) = \prod_{A \in U} P(A \mid \mathrm{pa}(A)) = P(U)$$

Local consistency



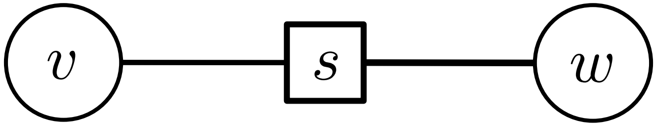
- The edge v—w is (locally) *consistent*, if marginalization from either v or w yields the same potential $\psi(s)$,

$$\sum_{v \setminus s} \psi(v) = \psi(s) = \sum_{w \setminus s} \psi(w)$$

- Our goal is to modify the potentials in such a way that consistency is maintained and the potentials are exactly the marginals of P(U),

$$\sum_{v \setminus s} \psi(v) = P(s) = \sum_{w \setminus s} \psi(w)$$

Absorption



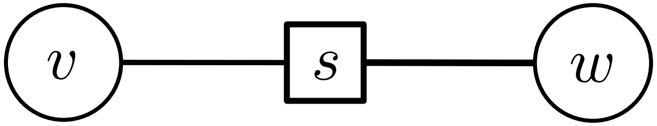
- Suppose we have modified ψ(v) to a new potential ψ*(v).
- *Absorption* replaces ψ(s) by ψ*(s) and ψ(w) by ψ*(w) as

$$\psi^*(s) = \sum_{v \setminus s} \psi^*(v), \qquad \psi^*(w) = \psi(w) \frac{\psi^*(s)}{\psi(s)}$$

to maintain local consistency:

$$\sum_{w \setminus s} \psi^*(w) = \frac{\psi^*(s)}{\psi(s)} \sum_{w \setminus s} \psi(w) = \frac{\psi^*(s)}{\psi(s)} \psi(s) = \sum_{v \setminus s} \psi^*(v)$$

Supportiveness

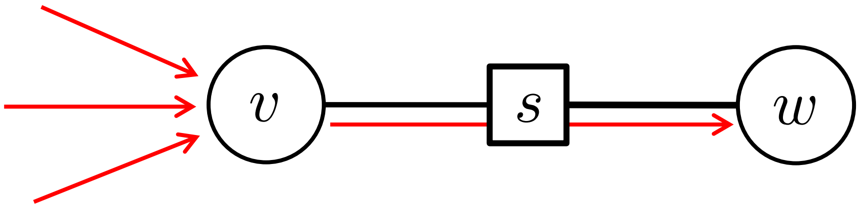


- A *supportive* edge allows absorption in both directions.
- The probability distribution represented by a supportive cluster tree is invariant under absorption, because

$$\frac{\psi^*(w)}{\psi^*(s)} = \frac{\psi(w)\psi^*(s)}{\psi^*(s)\psi(s)} = \frac{\psi(w)}{\psi(s)}$$

- Thus, we can manipulate the cluster tree in a series of absorptions while maintaining the representation of P(U).

Message passing (belief propagation)



- Absorption on v—w is passing a message from v to w via s.
- Message passing algorithm
  - Each node v sends a message to its neighbor if it has received messages from all other neighbors.
- After message passing on a supportive cluster tree
  - there will be no further changes in the potentials (convergence)
  - each link is consistent

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D-BSSE
Department of Biosystems
Science and Engineering

Global consistency

- If A $\in$ v and A $\in$ w, then

$$\sum_{v \setminus A} \psi(v) = \sum_{w \setminus A} \psi(w)$$

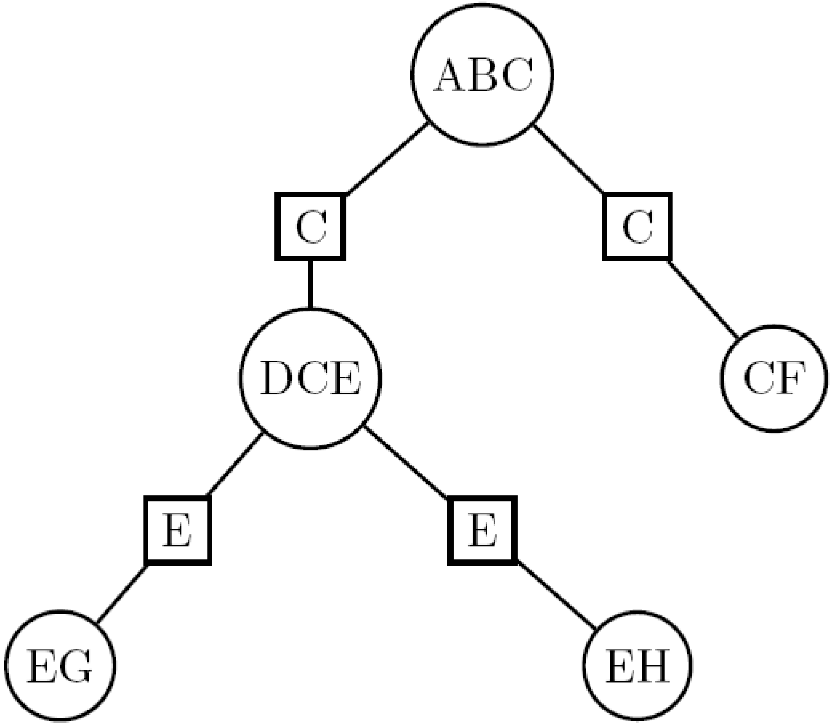
is guaranteed to hold only if v and w are neighbors (local consistency).

- A locally consistent cluster tree is *globally consistent*, if for **all** pairs of nodes v, w with intersection v $\cap$ w = I,

$$\sum_{v \setminus I} \psi(v) = \sum_{w \setminus I} \psi(w)$$

Junction tree

- A cluster tree is a *junction tree*, if for each pair of nodes (v, w), all nodes on the path between v and w contain the intersection v ∩ w (*running intersection property*).
- A locally consistent junction tree is globally consistent.

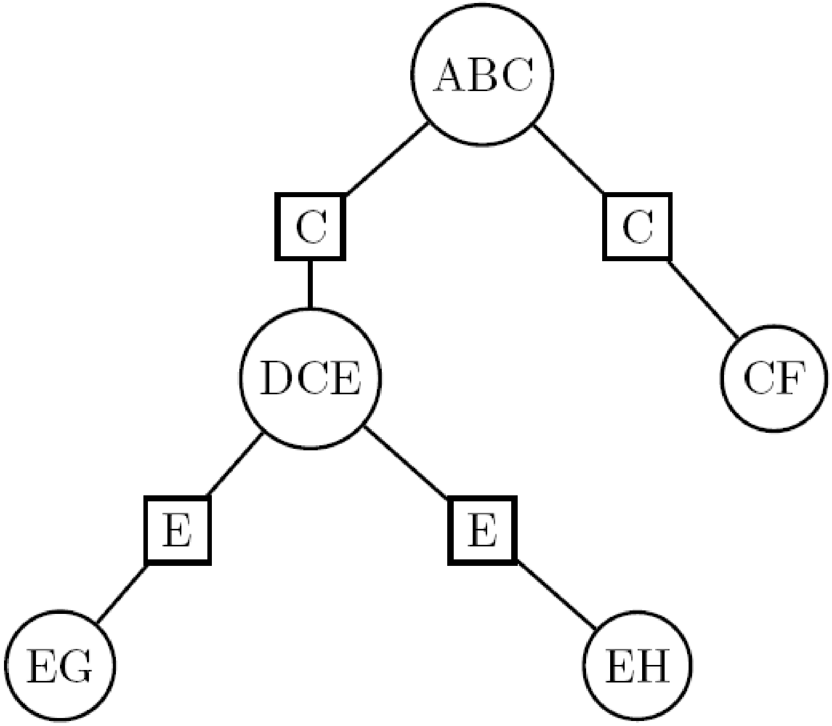


Potential marginals

- After message passing on the junction tree

$$\psi(v) = \sum_{U \setminus v} \psi(U)$$

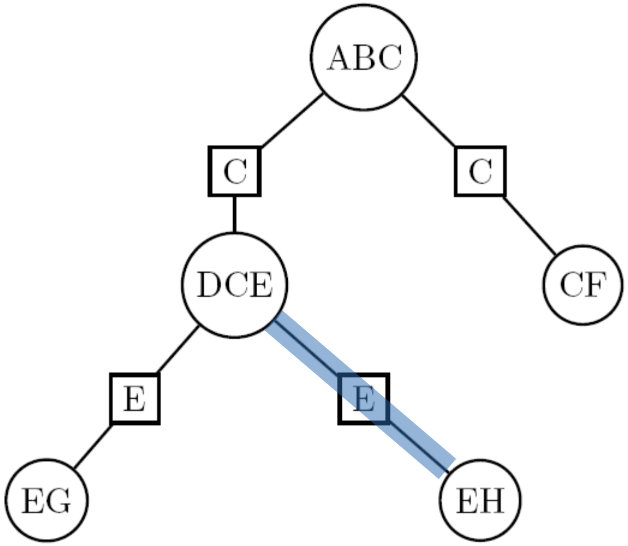
for each cluster v, where ψ(U) is the product of cluster potentials divided by the product of separator potentials.



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

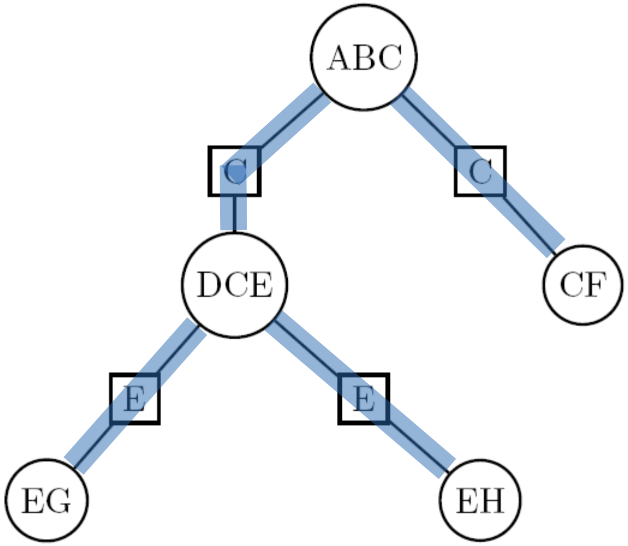
D-BSSE
Department of Biosystems
Science and Engineering

Example



$$
\begin{aligned}
P(A,B,C) &= \sum_{DEFGH} P(U) = \sum_{DEFGH} \psi(U) \\
&= \sum_{DEFGH} \frac{\psi(ABC)\psi(DCE)\psi(CF)\psi(EG)\psi(EH)}{\psi(C)^2\psi(E)^2} \\
&= \sum_{DEFG} \frac{\psi(ABC)\psi(DCE)\psi(CF)\psi(EG)}{\psi(C)^2\psi(E)} \underbrace{\sum_{H} \frac{\psi(EH)}{\psi(E)}}_{=1}
\end{aligned}
$$

Example



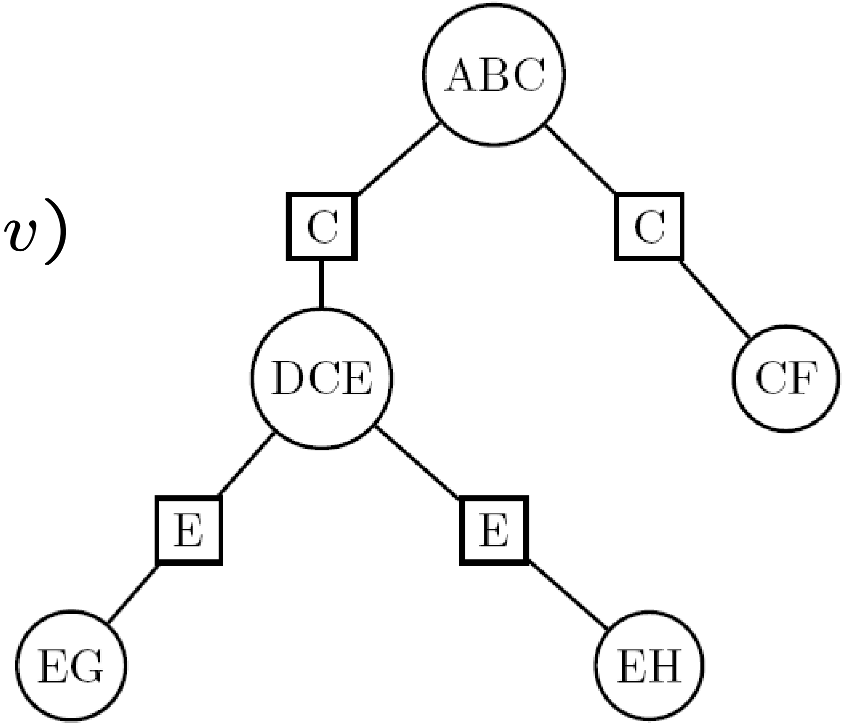
$$
\begin{aligned}
P(A,B,C) &= \sum_{DEFG} \frac{\psi(ABC)\psi(DCE)\psi(CF)\psi(EG)}{\psi(C)^2\psi(E)} \\
&= \sum_{DEFG} \frac{\psi(ABC)\psi(DCE)}{\psi(C)} \frac{\psi(CF)}{\psi(C)} \frac{\psi(EG)}{\psi(E)} \\
&= \sum_{DE} \psi(ABC)\frac{\psi(DCE)}{\psi(C)} = \psi(ABC)
\end{aligned}
$$

Computing the marginals of P(U)

- After message passing on the junction tree,

$$\begin{aligned}\psi(v) &= \sum_{U \setminus v} \psi(U) = P(v) \\ \psi(s) &= P(s)\end{aligned}$$

- Within clusters, we have to marginalize by brute force summation.
- Thus, computational complexity is exponential in maximal cluster size.

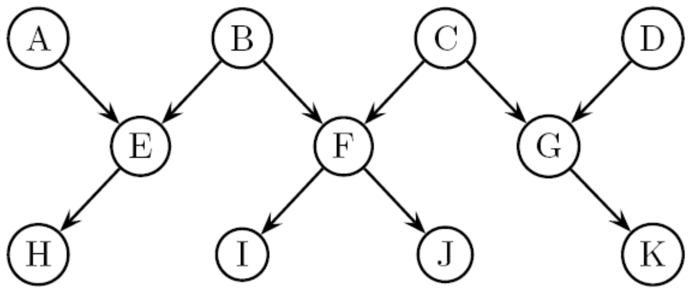


Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

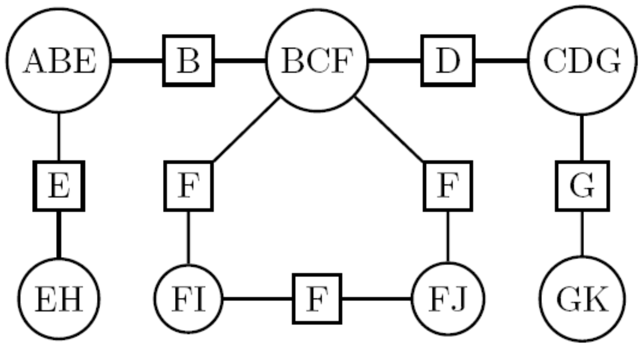
D-BSSE
Department of Biosystems
Science and Engineering

Constructing junction trees of singly connected graphs

1. For each variable A, form the cluster v = {A} ∪ pa(A).
2. Between any two intersecting clusters v and w add an edge with the separator $s = v \cap w \neq \emptyset$.
3. Each cycle in the resulting junction graph has the same separator and can therefore be broken.



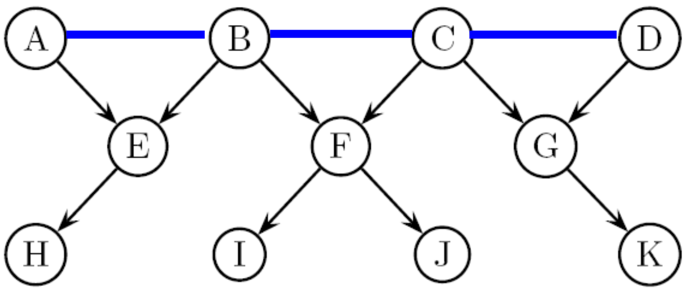
DAG



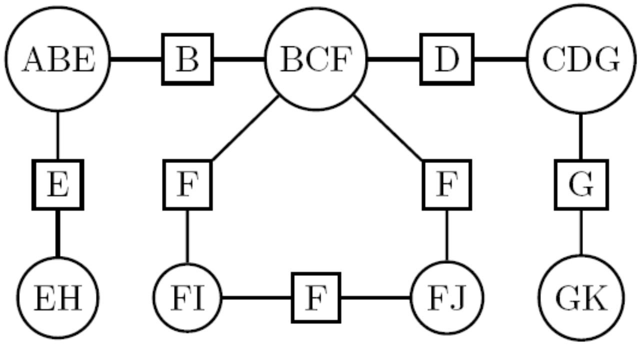
Junction graph

Moralization

- We can illustrate the condition v = {A} ∪ pa(A) by connecting the parents of A in the DAG, i.e., by moralizing the DAG.
- Then, the clusters are exactly the cliques in the moralized graph.

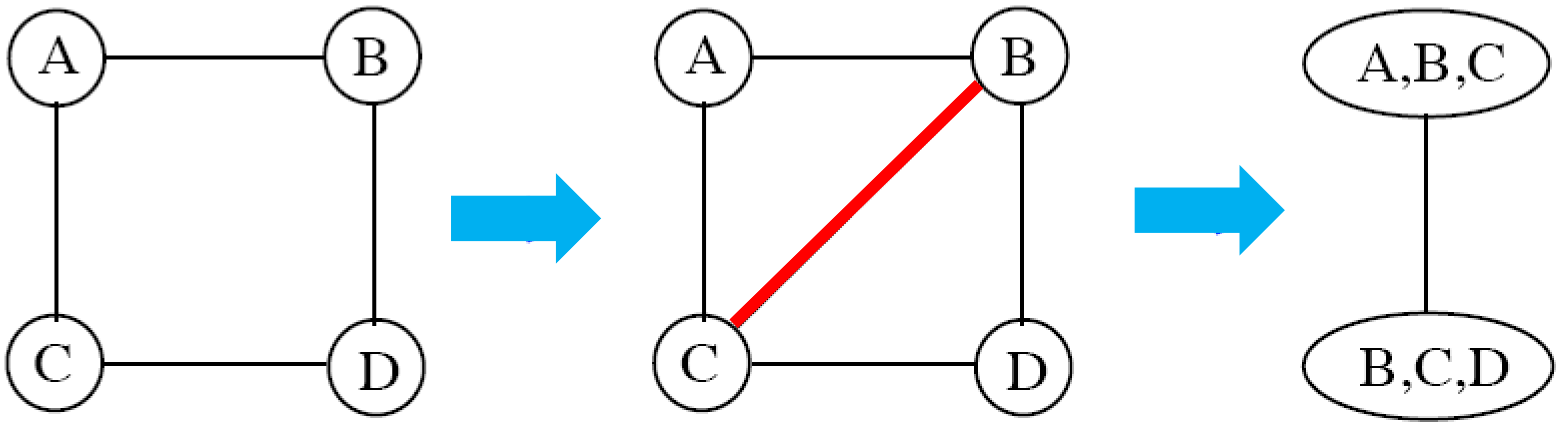


Moralized DAG



Junction graph

Constructing junction trees of loopy graphs



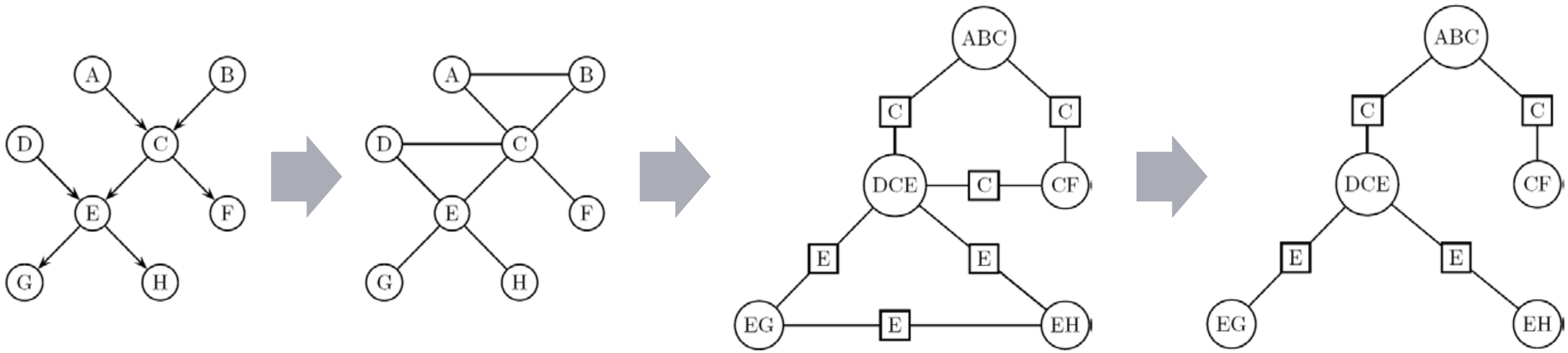
DAG
clique size = 2,
clique graph does not satisfy the running intersection property

Triangulated DAG
clique size = 3

Clique graph of triangulated DAG
satisfies the running intersection property

Theorem: An undirected graph is triangulated, if and only if its junction graph is a junction tree.

Junction tree algorithm



DAG Moral graph Junction graph Junction tree

1. Moralize the DAG
2. Triangulate the DAG
3. Build clique graph of the DAG
4. Break cycles in clique graph
5. Assign potentials
6. Pass messages
7. Compute clique marginals
8. Marginalize within cliques

Remarks

- The computational complexity of marginalization is exponential in the clique size.
- Finding a triangulation with minimal clique size is an NP-hard problem, but good heuristics exist.
- Undirected graphical models (Markov random fields, MRF) can also be handled without the need for moralization.
- The most probable configuration (MAP estimate) of a subset of variables can be found in a similar fashion.

Summary

- Factor graphs can represent both directed and undirected graphical models.
- Exact inference in acyclic graphical models can be performed efficiently using the sum-product algorithm (message passing, belief propagation).
- Exact inference in general graphical models can be performed efficiently (in the maximal clique size) using the junction tree algorithm, a direct generalization of the sum-product algorithm.
- Generalization:

  forward-backward < sum-product < junction tree

D-BSSE
Department of Biosystems
Science and Engineering

References

- Barber D. Probabilistic Modelling and Reasoning: The Junction Tree Algorithm. Course notes, 2003–2004.
- Huang C, Darwiche A (1996). Inference in belief networks: A procedural guide. *Int J Approx Reason* 15:225–263.
- Jordan MI (ed.). Learning in Graphical Models. Kluwer Academic Publishers, 1998.
- Bishop CM. Pattern Recognition and Machine Learning. Chapter 8.