# Bayesian Networks

# 5. Exact Inference / Clustering

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Business Economics and Information Systems
& Institute for Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

## Components and cycles

**Definition 1.** Let $G$ be an undirected graph. $G$ is called **connected**, if there is a path from any vertex to any other vertex:

$$G^*(v,w) \neq \emptyset, \quad \forall v,w \in V$$

For a vertex $v \in V$ we call

$$\text{comp}_G(v) := \{w \,|\, G^*(v,w) \neq \emptyset\}$$

the **(connection) component of** $v$ **in** $G$.

A proper path $p = (v_1, \ldots, v_n)$ is called **cyclic**, if $v_1 = v_n$ and $v_i$ are pairwise different otherwise:

$$v_i = v_j \Leftrightarrow i = 1 \text{ and } j = n$$

A proper path $p = (v_1, \ldots, v_n)$ is called **simple**, if $v_i$ are pairwise different.

An undirected graph $G$ is called **acyclic**, if it does not contain a cyclic path.

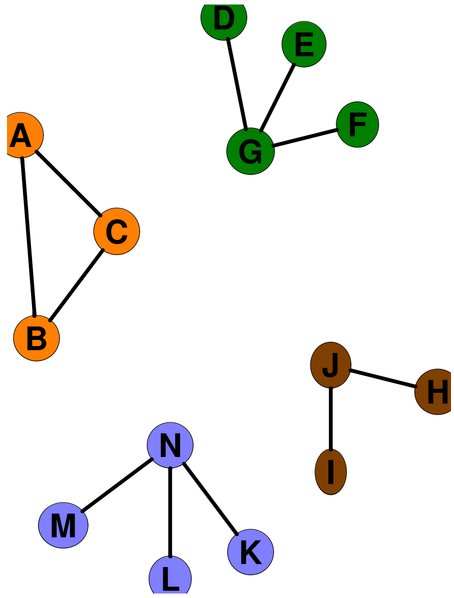



Figure 1: Graph with four components (colored).



Bayesian Networks / 1. Trees

## Trees

**Definition 2.** An undirected graph $G$ is called **unrooted/undirected tree**, if

(i) it is connected and acyclic

*or equivalently*

(ii) there is exactly one simple path between any two vertices:

$$|G^*_{\text{simple}}(v,w)| = 1, \quad \forall v,w \in V$$

The unique simple path between $v$ and $w$ is denoted by $\text{path}_G(v,w)$.

A directed graph $G$ is called **(rooted/directed) tree**, if every vertex but one (called **root**) has exactly one parent and the root has no parents:

$$\exists r \in V : \text{pa}(r) = \emptyset \text{ and } \forall v \in V, v \neq r : |\text{pa}(v)| = 1$$

Rooted trees are special DAGs.

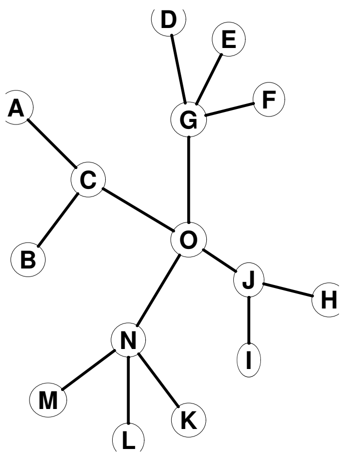



Figure 2: An unrooted tree.

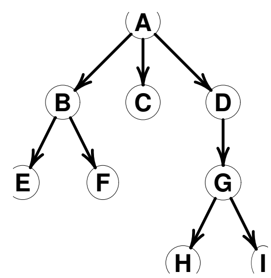



Figure 3: A (rooted) tree.

## Trees / leaves

**Definition 3.** Let $G = (V, E)$ be an unrooted tree and $r \in V$ any vertex. Then the directed graph $\text{tree}(G, r) := (V, E')$ with

$$E' := \{(v, w) \mid \{v, w\} \in E, |\text{path}(r, v)| < |\text{path}(r, w)|\}$$

is called **tree rooted at** $r$ **of** $G$. Obviously the tree rooted at $r$ is a rooted tree with root $r$.

For an unrooted tree the vertices with only one neighbor are called **leaves.**

For a rooted tree vertices other than the root with only one neighbor are called **leaves**; the root is called a **leaf** if it is the only vertex of the tree.

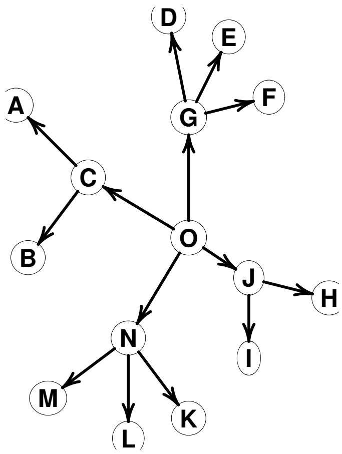



Figure 4: Unrooted tree from figure 2 rootet at $O$.

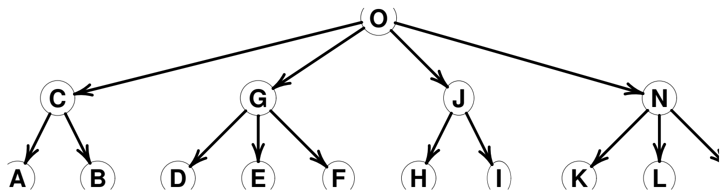



Figure 5: The same tree as in figure 4





## Trees / level maps

**Definition 4.** Let $G$ be a DAG (e.g., a rooted tree). The length of the longest path is called the **depth of** $G$ and denoted by $\text{depth}(G)$.

Let $G := (V, E)$ be a DAG (e.g., a rooted tree). A map

$$\lambda : V \to \mathbb{N}$$

is called **level map of** $G$ if

$$\lambda(v) > \lambda(\text{pa}(v)), \quad \forall v \in V$$

For a rooted tree $G := (V, E)$ with root $r$,

$$\text{depth}(v) := |\text{path}(r, v)|$$

and

$$\text{height}(v) := \text{depth}(G) - \max\{|p| \mid w \in V \text{ leaf}, p \in G^*(v, w), r \notin p\} + 1$$

are examples for level maps.

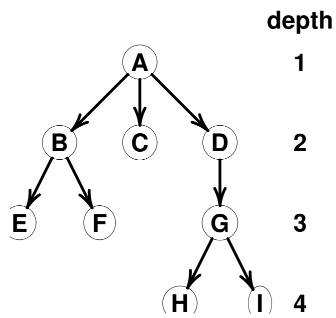



Figure 6: The depth level map for a tree.

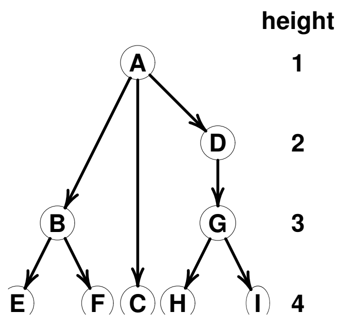



Figure 7: The height level map for a tree.

## Links, polytrees

**Definition 5.** Let $G := (V, E)$ be an undirected graph. The set

$$L_G := \{(v, w) \mid \{v, w\} \in E\}$$

is called its **set of links**.

**Definition 6.** A directed graph $G$ is called **polytree**, if for each vertex $r$ without parents (called a root) its descendants $\operatorname{desc} r \cup \{r\}$ form a tree.

*or equivalently*

if every vertex has at most one parent that is not a root (i.e., has parents itself).

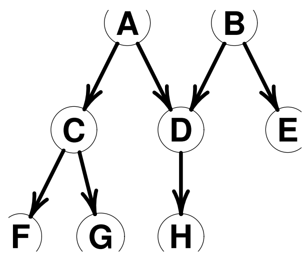



Figure 8: A polytree with roots $A$ and $B$.

# Cluster trees

**Definition 7.** Let $V$ be a set (of variables).
An unrooted tree $G := (\mathcal{V}, E)$ on $\mathcal{V} \subseteq \mathcal{P}(V)$ is called a **cluster tree on** $V$, if

(i) the induced subgraph on all vertices containing a given variable $v$, i.e.,

$$\{W \in \mathcal{V} \,|\, v \in W\}$$

is connected for all variables $v \in V$.

*or equivalently*

(ii) for any $U, W \in \mathcal{V}$:

$$U \cap W = U \cap \bigcup \text{comp}_{G \setminus \{U\}}(W)$$

For two vertices $U, W$ of a cluster tree $U \cap W$ is called their **separator**.

Cluster trees are also called **join trees** and **junction trees**.

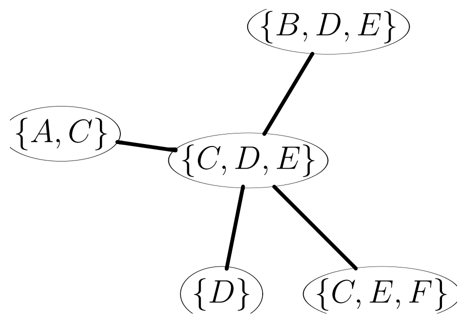



Figure 9: A cluster tree on $V := \{A, B, C, D, E, F\}$.

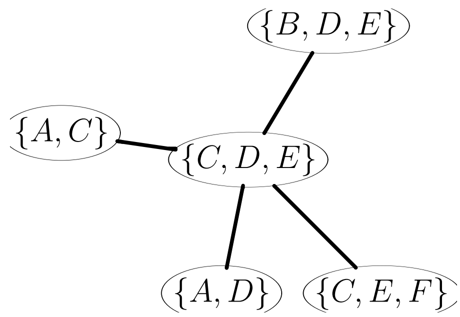



Figure 10: Not a cluster tree.





# Cluster trees

**Definition 8.** Let $V$ be a set of variables and $Q$ be a set of potentials on $V$.
A cluster tree $G := (\mathcal{V}, E)$ on $V$ with a map

$$Q_G : \mathcal{V} \to \mathcal{P}(Q)$$

s.t.

(i) $\text{dom}(q) \subseteq C$ for all $q \in Q_G(C)$, $C \in \mathcal{V}$,

(ii) $\text{Im}(Q_G)$ covers $Q$, i.e.,

$$\bigcup_{W \in \mathcal{V}} Q_G(W) = Q$$

and

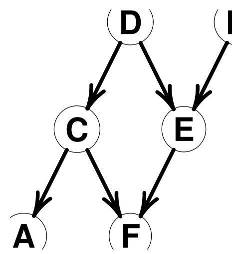
(iii) $Q_G(W)$ and $Q_G(U)$ are pairwise disjunct, i.e.,

$$Q_G(W) \cap Q_G(U) \neq \emptyset \Rightarrow W = U, \quad \forall W, U \in \mathcal{V}$$

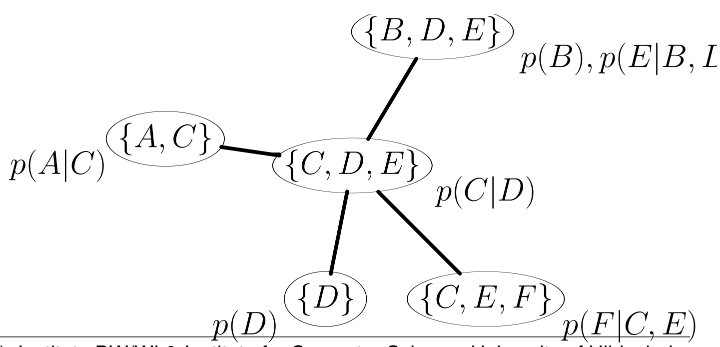
is called **a cluster tree for Q**.

$$Q := \{p(D), p(B), p(C|D), p(E|D, B), p(A|C), p(F|C, E)\}$$

are the conditional probabilities of the bayesian network




A cluster tree for $Q$ is, e.g.,



## A simple cluster tree for polytree Bayesian networks

Let $G$ be a directed graph. For $v \in V$

$$\text{fam}(v) := \{v\} \cup \text{pa}(v)$$

is called the **familiy of** $v$.

Let $(G = (V, E), (p_v)_{v \in V})$ be a polytree Bayesian network. Let

$$\mathcal{V} := \{\text{fam}(v) \,|\, v \in V\}$$

and

$$F := \{(\text{fam}(\text{pa}(v)), \text{fam}(v)) \,|\, v \in V, \text{pa}(v) \neq \emptyset\}$$

Then $H := (\mathcal{V}, F)$ is a cluster tree for $Q := \{p_v \,|\, v \in V\}$ called **family tree**.

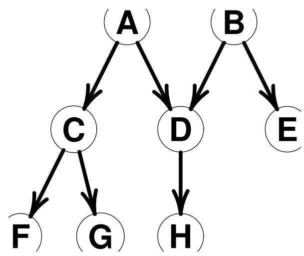



Figure 11: Polytree Bayesian network.

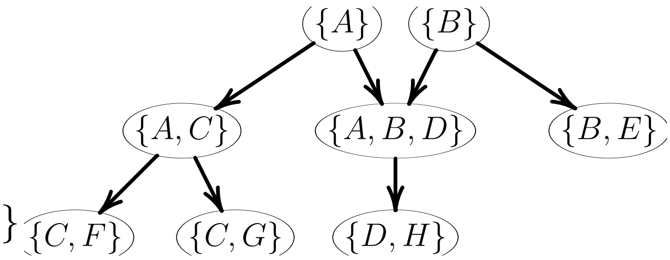



Figure 12: Cluster tree of polytree Bayesian network above.





## Clique cluster tree for Markov networks

Markov networks $(G, (q_C)_{C \in \mathcal{C}(G)})$ use potentials on cliques to specify the JPD. If $G$ is triangulated, it allows a chain of cliques, i.e., an ordering $C_1, \ldots, C_n$ of the cliques that satisfies the running intersection property:

$$C_i \cap \bigcup_{j<i} C_j \subseteq C_{k(i)}, \quad \forall i \exists k(i) < i$$

We can construct the **clique (cluster) tree** $H := (\mathcal{V}, F)$ from

$$\mathcal{V} := \mathcal{C}(G) = \{C_1, \ldots, C_n\}$$

and

$$F := \{(C_{k(i)}, C_i) \,|\, i = 2, \ldots, n\}$$

We will later address the problem of cluster trees for non-triangulated Markov networks.

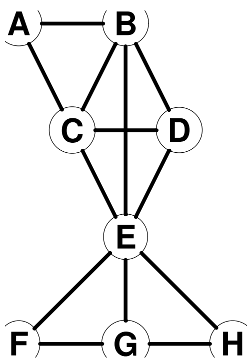



Figure 13: Markov network.

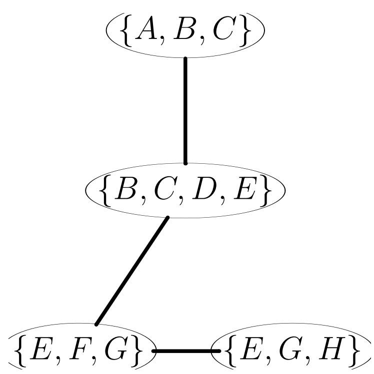



Figure 14: Clique cluster tree of Markov network above.

## Clique cluster tree for Bayesian networks

Cluster trees for Bayesian networks can be constructed by a two phase approach:

(i) construct an equivalent Markov network representation of the Bayesian network,

(ii) construct the clique cluster tree for the Markov network.

An equivalent Markov network for a Bayesian network $(G = (V, E), (p_v)_{v \in V})$ can be constructed by

$$\text{moral}(G)$$

and assigning the conditional probabilities to cliques that contain their domain.

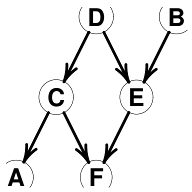



Figure 15: Bayesian network.

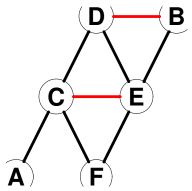



Figure 16: Markov network for Bayesian network above.

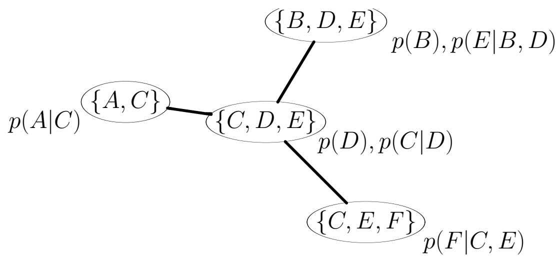



Figure 17: Clique cluster tree for Markov network above.

## Vertex marginals and link potentials

Let $Q$ be a set of potentials and $G$ be a cluster tree for $Q$.
Inference for all variables separately can be accomplished by

(i) adding the evidence potentials to $Q$ (and to $Q_G$),

(ii) computing the **vertex marginals**

$$q_V := (\prod_{q \in Q} q)^{\downarrow V}$$

(iii) computing the single variable marginals

$$q_v := (q_V)^{\downarrow v}, \quad \text{for } V \in \mathcal{V} \text{ with } v \in V$$

This can be done by a recursive computation of the **link potentials**:

$$q_{U,W} := (\prod_{q \in Q_G(\text{comp}_{G \setminus \{W\}}(U))} q)^{\downarrow U \cap W}$$

traditionally called **messages**.

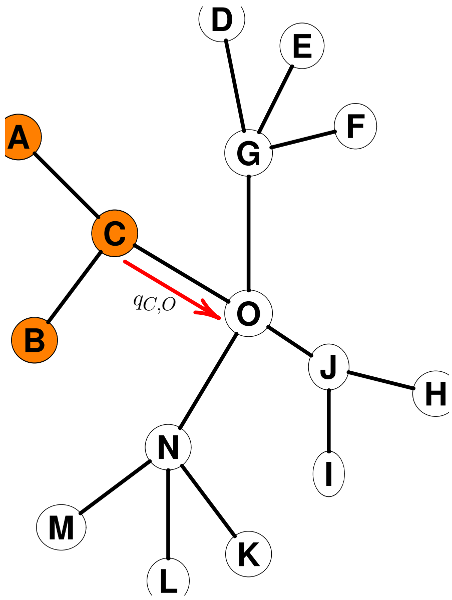



Figure 18: The link potential $q_{C,O}$ describes the potentials in the component $\text{comp}_{G \setminus \{O\}}(C)$ (orange).





## Link potentials

**Lemma 1.** *Vertex marginals and link potentials can be expressed by link potentials:*

*(i)*

$$q_U = \prod_{q \in Q_G(U)} q \prod_{T \in \text{fan}(U)} q_{T,U}$$

*(ii)*

$$q_{U,W} = (\prod_{q \in Q_G(U)} q \prod_{\substack{T \in \text{fan}(U), \\ T \neq W}} q_{T,U})^{\downarrow U \cap W}$$

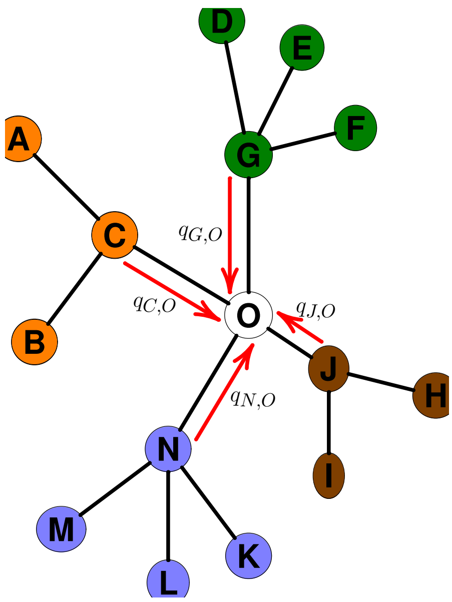



Figure 19: Expressing the vertex potential $q_O$ by the linkpotentials $q_{\cdot,O}$.

## Link potentials

**Lemma 1.** *Vertex marginals and link potentials can be expressed by link potentials:*

*(i)*

$$q_U = \prod_{q \in Q_G(U)} q \prod_{T \in \text{fan}(U)} q_{T,U}$$

*(ii)*

$$q_{U,W} = \Big( \prod_{q \in Q_G(U)} q \prod_{\substack{T \in \text{fan}(U),\\ T \neq W}} q_{T,U} \Big)^{\downarrow U \cap W}$$

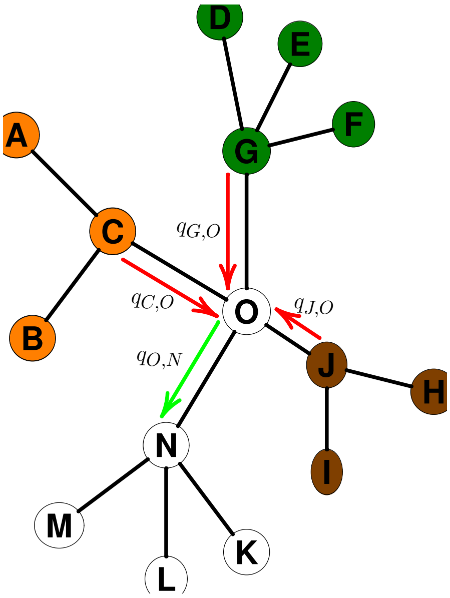



Figure 20: Expressing the link potential $q_{O,N}$ by the linkpotentials $q_{\cdot,O}$.





## Recursive computation of link potentials

**Lemma 2.** *The formula of the previous lemma allows the recursive computation of link potentials in a cluster tree $G$.*

*Proof.* Choose an arbitrary vertex as root and replace $G$ by its rooted tree.
Let $\lambda$ be a level map of $G$ and $\lambda_{\min}, \lambda_{\max}$ its minimal and maximal values.
I. up links (**collect evidence**): induction on $n := \lambda(U)$ for link potentials $q_{U,\text{pa}(U)}$.

$n = \lambda_{\max}$: $U$ is a leaf and has no other neighbors other than its parent.

$n \rightarrow n-1$: the link potentials from childs into $U$ have already been computed by induction hypothesis. $\Rightarrow$ $q_{U,\text{pa}(U)}$ can be computed ($G$ is a tree, thus $U$ has at most one parent). □

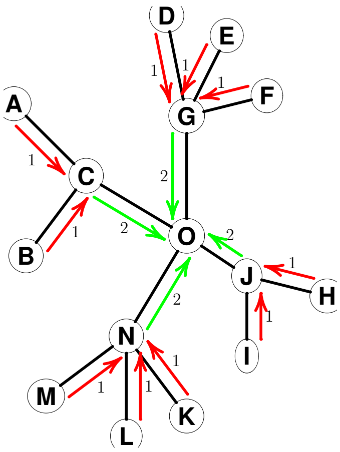



Figure 21: Collect evidence.

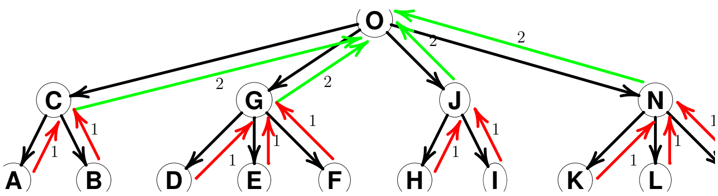



Figure 22: Collect evidence.

## Recursive computation of link potentials

**Lemma 2.** *The formula of the previous lemma allows the recursive computation of link potentials in a cluster tree $G$.*

*Proof (cont.).*
II. down links (**distribute evidence**): induction on $n := \lambda(\mathrm{pa}(U))$ for link potentials $q_{\mathrm{pa}(U),U}$.

$n = \lambda_{\min}$: $\mathrm{pa}(U)$ is the root. All of its neighboring link potentials have been computed by step I. $\Rightarrow$ $q_{\mathrm{pa}(U),U}$ can be computed.

$n \rightarrow n+1$: the link potentials from childs into $\mathrm{pa}(U)$ have already been computed by step I, the link potential $q_{\mathrm{pa}(\mathrm{pa}(U)),\mathrm{pa}(U)}$ has already been computed by induction hypothesis. $\Rightarrow$ $q_{\mathrm{pa}(U),U}$ can be computed.

□

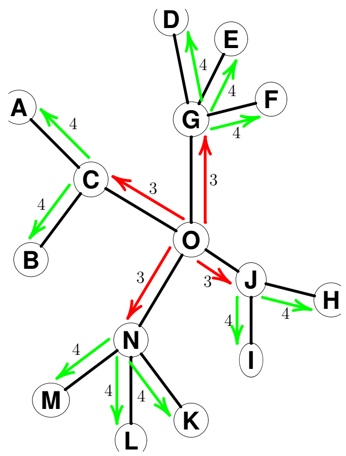



Figure 23: Distribute evidence.

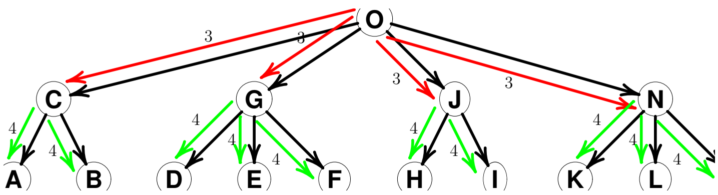



Figure 24: Distribute evidence.





## Shafer-Shenoy propagation

The following computation scheme is called **Shafer-Shenoy propagation** []:

(i) collect evidence:

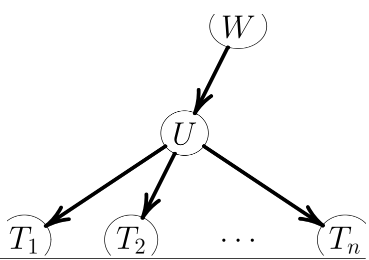
$$q_{U,W} = \Big(\prod_{q\in Q_G(U)} q \prod_{\substack{T\in\mathrm{fan}(U),\\ T\neq W}} q_{T,U}\Big)^{\downarrow U\cap W} = \Big(\Big(\prod_{q\in Q_G(U)} q\Big)\cdot q_{T_1,U}\cdots q_{T_n,U}\Big)^{\downarrow U\cap W}$$

(ii) distribute evidence:

$$q_{U,T_i} = \Big(\prod_{q\in Q_G(U)} q \prod_{\substack{T\in\mathrm{fan}(U),\\ T\neq T_i}} q_{T,U}\Big)^{\downarrow U\cap T_i} = \Big(\Big(\prod_{q\in Q_G(U)} q\Big)\cdot q_{W,U}\cdot q_{T_1,U}\cdots \widehat{q_{T_i,U}}\cdots q_{T_n,U}\Big)^{\downarrow U\cap T_i}$$

(iii) marginalize:

$$q_U = \prod_{q\in Q_G(U)} q \prod_{T\in\mathrm{fan}(U)} q_{T,U} = \Big(\prod_{q\in Q_G(U)} q\Big)\cdot q_{W,U}\cdot q_{T_1,U}\cdots q_{T_n,U}$$



## Hugin propagation

The following computation scheme is called **Hugin propagation** []:

(i) collect evidence:

$$q'_U = \prod_{q \in Q_G(U)} q \prod_{\substack{T \in \text{fan}(U) \\ T \neq W}} q_{T,U} = (\prod_{q \in Q_G(U)} q) \cdot q_{T_1,U} \cdots q_{T_n,U}$$

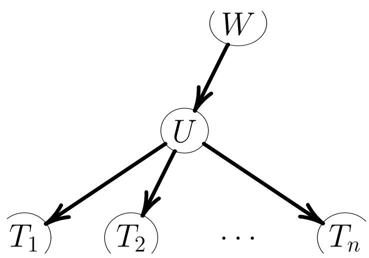
$$q_{U,W} = q_U'^{\downarrow U \cap W}$$

(ii) marginalize and distribute evidence:

$$q_U = q'_U \cdot q_{W,U}$$

$$q_{U,T_i} = (\frac{q_U}{q_{T_i,U}})^{\downarrow U \cap T_i}$$

but store separator marginal $(q_U)^{\downarrow U \cap T_i}$








## Shafer-Shenoy vs. Hugin propagation

Hugin propagation compared to Shafer-Shenoy propagation:

(i) Hugin propagation allows the reuse of the storage space of the link potentials $q_{U,W}$ for $q_{W,U}$ (one "postbox" instead of two),

(ii) Hugin propagation affords extra storage space for the vertex potentials $q_U$ and thus its overall space requirements are higher,

(iii) Hugin propagation requires a smaller number of total operations (additions, multiplications, divisions) than Shafer-Shenoy propagation at vertices with degree $> 3$ (that can be avoided by the use of binary cluster trees),

(iv) Hugin propagation allows the marginalization of the smaller separator marginals,

(v) Some of the operations required by Hugin propagation are more costly (divisions) than those required by Shafer-Shenoy.

## Lazy propagation

The idea of **lazy propagation** [MJ98] is to keep the link potentials in factored form, i.e., to replace the link potential $q_{U,W}$ with a set of potentials $Q_{U,W}$ with

$$q_{U,W} = \prod_{q \in Q_{U,W}} q$$

The formulas of lemma 1 then read as:

(i)

$$q_U = \prod_{q \in Q_G(U)} q \prod_{\substack{T \in \mathrm{fan}(U) \\ q \in Q(T,U)}} q$$

(ii)

$$q_{U,W} = \mathrm{elim}(Q_G(U) \cup \bigcup_{\substack{T \in \mathrm{fan}(U), \\ T \neq W}} Q_{T,U}, c(U \cap W))$$

## Clique trees for triangulated graphs (1/3)

Clique cluster trees can easily be computed of triangulated graphs.

(i) Triangulated graphs admit a perfect ordering of $G$, i.e., an ordering $\sigma$ with

$$\mathrm{fam}_{\sigma(\{1,\ldots,i\})}(\sigma(i))$$

is complete.

(ii) A perfect ordering can be computed by the maximum cardinality search algorithm (MCS).

Proving the correctness of MCS affords some work (e.g., [Sha94, p. 43–46]).

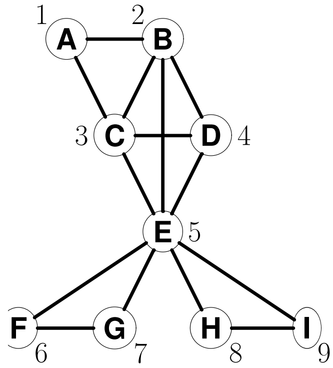



Figure 25: Perfect ordering of a triangulated graph obtained by MCS.

perfect-ordering-MCS$(G = (V, E))$ :
**for** $i = 1, \ldots, |V|$ **do**
    $\sigma(i) := v \in V \setminus \sigma(\{1, \ldots, i-1\})$ with maximal $|\mathrm{fan}_G(v) \cap \sigma(\{1, \ldots, i-1\})|$
        breaking ties arbitrarily
**od**
**return** $\sigma$

Figure 26: MCS algorithm to compute a perfect ordering [TY84].





## Clique trees for triangulated graphs (2/3)

All cliques can be enumerated by a variant of the MCS algorithm:

1. if $G$ is triangulated, MCS computes a perfect ordering of $G$, i.e., $\mathrm{fam}_{\sigma(\{1,\ldots,i\})}(\sigma(i))$ is complete.
2. we get all cliques this way, as for each clique $C$ let $i := \max \sigma^{-1}(C)$, then $C = \mathrm{fam}_{\sigma(\{1,\ldots,i\})}(\sigma(i))$.

enumerate-cliques-MCS$(G = (V, E))$ :
$\mathcal{C} := \emptyset$
**for** $i = 1, \ldots, |V|$ **do**
    $\sigma(i) := v \in V \setminus \sigma(\{1, \ldots, i-1\})$ with maximal $|\mathrm{fan}_G(v) \cap \sigma(\{1, \ldots, i-1\})|$
        breaking ties arbitrarily
    $\mathcal{C} := \mathcal{C} \cup \{\mathrm{fam}_{\sigma(\{1,\ldots,i\})}(\sigma(i))\}$
**od**
$\mathcal{C} := \{C \in \mathcal{C} \mid \nexists D \in \mathcal{C} : D \supsetneq C\}$
**return** $\mathcal{C}$

Figure 27: MCS algorithm to compute cliques of a triangulated graph [TY84].

Let $C_i := \mathrm{fam}_{\sigma(\{1,\ldots,i\})}(\sigma(i))$ and

$$C_i = \{\sigma(j_1), \ldots, \sigma(j_n), \sigma(i)\}$$

with $j_1 < j_2 < \ldots < j_n$. Due to the completeness of $C_i$ then $\sigma(j_n)$ is a neighbor of all $\sigma(j_l)$, $l = 1, \ldots, n-1$, and thus

$$C_i \cap \bigcup_{k<i} C_k \subseteq C_{j_n}$$

i.e., the sequence $(C_i)_{i=1,\ldots,|V|}$ has the running intersection property (that can be telescoped if a $C_i$ gets pruned).

## Clique trees for triangulated graphs (3/3)

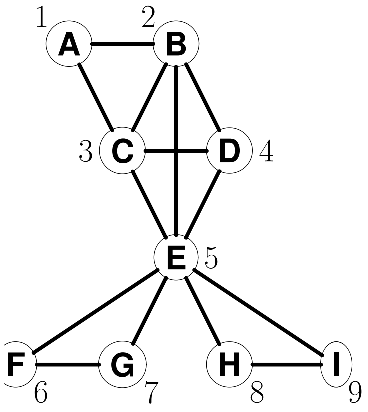



Figure 28: Perfect ordering of a triangulated graph obtained by MCS.

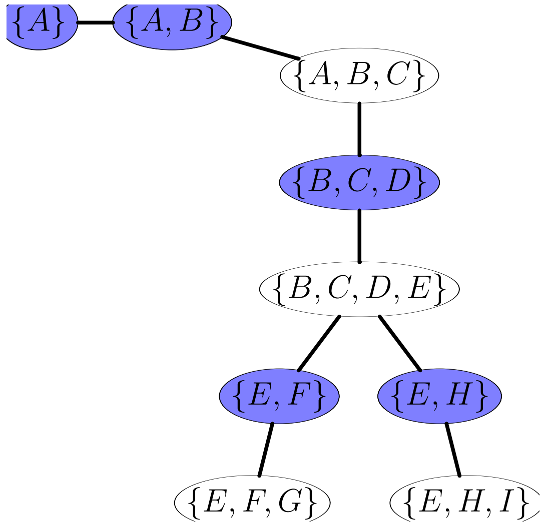



Figure 29: Clique cluster tree for triangulated graph at the left (blue nodes are temporary and pruned).





**1. Trees**

**2. Cluster Trees**

**3. Recursive Computation of Link Potentials**

**4. Clique (Cluster) Trees**

**5. Triangulation**

## Triangulation of graphs (1/3)

As clique cluster trees can easily be computed of triangulated graphs, we triangulate non-triangulated graphs by filling-in additional edges.

However, additional edges mean, that the graph represents a smaller portion of the independency statements, and thus, inference becomes harder.

The fewer edges have to be filled-in, the better.

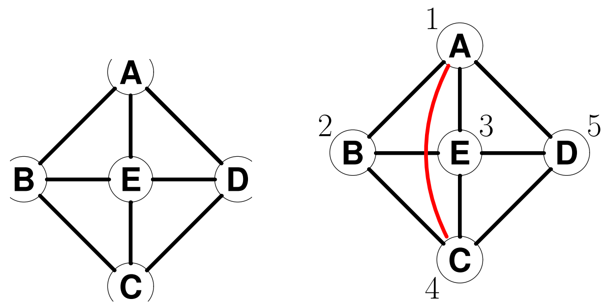



Figure 30: Non-triangulated graph and its triangulation obtained by MCS.

triangulate-MCS$(G = (V, E))$ :
$\sigma :=$ *perfect-ordering-MCS*$(G)$
*fillin* $:= \emptyset$
**for** $i = |V|, \ldots, 1$ **do**
    *fillin* $:=$ *fillin* $\cup \{(u, w) \mid u, w \in \text{fan}_{(V, E \cup \textit{fillin})}(\sigma(i)) \cap \sigma(\{1, \ldots, i-1\}), \{u, w\} \notin E\}$
**od**
**return** $G' := (V, E \cup \textit{fillin})$

Figure 31: Maximum cardinality search algorithm for triangulating a graph [TY84].





## Triangulation of graphs (2/3)

MCS does not guarantee to give best results (i.e., minimal fill-ins). It is just a heuristics that gives useable results (in most cases).

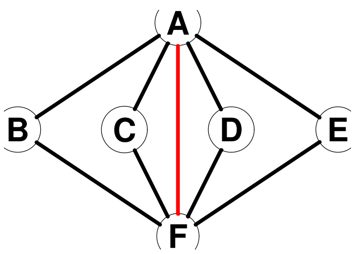



Figure 32: Optimal triangulation.

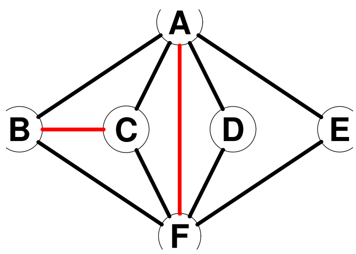



Figure 33: Non-optimal triangulation obtained by MCS (with smallest index rule).

# Triangulation of graphs (3/3)

Beneath the heuristic triangulation algorithms one distinguishes between:

**minimum triangulations:** no other triangulation has a smaller number of filled-in edges (global minimum).

This task is known to be NP-complete [Yan81].

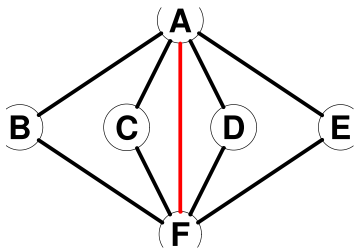



Figure 34: A minimum triangulation (here: unique).

**minimal triangulations:** no subset of the filled-in edges results in a triangulation (local minimum).

There are several algorithms for the minimal triangulation task, e.g., Lex-M [RTL76], MCS-M [BBH02], and LB-triang [BBH+03].

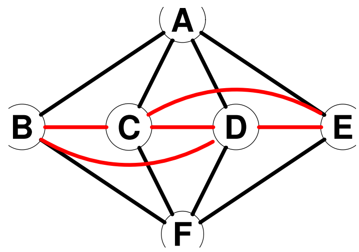



Figure 35: A minimal triangulation.





# References

[BBH02] A. Berry, J. R. S. Blair, , and P. Heggernes. Maximum cardinality search for computing minimal triangulations. In L. Kucera, editor, *Graph Theoretical Concepts in Computer Science, Proceedings of the 28th International Workshop on Graph Theoretical Concepts in Computer Science (WG 2002), Cesky Krumlov, Czech Republic, June 13-15*, 2002.

[BBH+03] A. Berry, J. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering, 2003.

[MJ98] Anders L. Madsen and Finn V. Jensen. Lazy propagation in junction trees. In *Proceedings of the 14th Conference on UAI*, pages 362–369, 1998.

[RTL76] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5:266–283, 1976.

[Sha94] Ron Shamir. Advanced topics in graph algorithms. Technical report, Tel-Aviv University, 1994.

[TY84] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduct acyclic hypergraphs. *SIAM Journal on Computing*, 13:566–579, 1984.

[Yan81] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM J. Alg. and Disc. Meth.*, 2:77–79, 1981.