EE512A – Advanced Inference in Graphical Models

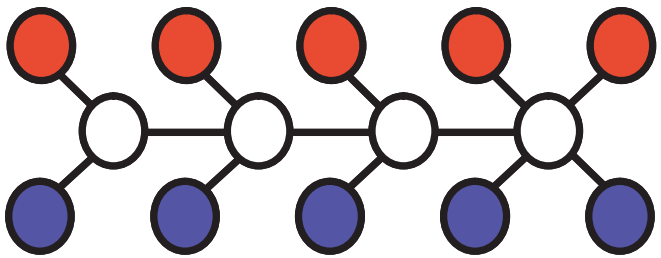
— Fall Quarter, Lecture 6 —

http://j.ee.washington.edu/~bilmes/classes/ee512a_fall_2014/

Prof. Jeff Bilmes

University of Washington, Seattle
Department of Electrical Engineering
http://melodi.ee.washington.edu/~bilmes

Oct 15th, 2014



Announcements

- Reading assignments, posted to our canvas announcements page (`https://canvas.uw.edu/courses/914697/announcements`): `intro.pdf`, `ugms.pdf` on undirected graphical models, and `tree_inference.pdf` on trees.
- Homework 1 is out, due next Tuesday (10/21) at 11:45pm, electronically via our assignment dropbox (`https://canvas.uw.edu/courses/914697/assignments`).

Class Road Map - EE512a

- L1 (9/29): Introduction, Families, Semantics
- L2 (10/1): MRFs, elimination, Inference on Trees
- L3 (10/6): Tree inference, message passing, more general queries, non-tree)
- L4 (10/8): Non-trees, perfect elimination, triangulated graphs
- L5 (10/13): triangulated graphs, the triangulation process/heuristics
- L6 (10/15): multiple queries, junction trees, intersection graphs,
- L7 (10/20):
- L8 (10/22):
- L9 (10/27):
- L10 (10/29):
- L11 (11/3):
- L12 (11/5):
- L13 (11/10):
- L14 (11/12):
- L15 (11/17):
- L16 (11/19):
- L17 (11/24):
- L18 (11/26):
- L19 (12/1):
- L20 (12/3):
- Final Presentations: (12/10):

Finals Week: Dec 8th-12th, 2014.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.
- Triangulated graph iff perfect elimination graph.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.
- Triangulated graph iff perfect elimination graph.
- All minimal triangulations of a graph can be created using elimination.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.
- Triangulated graph iff perfect elimination graph.
- All minimal triangulations of a graph can be created using elimination.
- k-trees, generalization of trees. Sometimes called hyper-tree. All min-separators are k-cliques. partial k-trees. Embedding into k-trees.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.
- Triangulated graph iff perfect elimination graph.
- All minimal triangulations of a graph can be created using elimination.
- k-trees, generalization of trees. Sometimes called hyper-tree. All min-separators are k-cliques. partial k-trees. Embedding into k-trees.
- Any triangulated graph G' can be embedded into k-tree where $k+1$ is the size of the largest clique of G'. Thus any graph can be embedded into a k-tree for large enough k.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.
- Triangulated graph iff perfect elimination graph.
- All minimal triangulations of a graph can be created using elimination.
- k-trees, generalization of trees. Sometimes called hyper-tree. All min-separators are k-cliques. partial k-trees. Embedding into k-trees.
- Any triangulated graph G' can be embedded into k-tree where $k+1$ is the size of the largest clique of G'. Thus any graph can be embedded into a k-tree for large enough k.
- NP-complete: finding smallest k such that G is embeddable into k-tree, inapproximability results.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.
- Triangulated graph iff perfect elimination graph.
- All minimal triangulations of a graph can be created using elimination.
- k-trees, generalization of trees. Sometimes called hyper-tree. All min-separators are k-cliques. partial k-trees. Embedding into k-trees.
- Any triangulated graph G' can be embedded into k-tree where $k+1$ is the size of the largest clique of G'. Thus any graph can be embedded into a k-tree for large enough k.
- NP-complete: finding smallest k such that G is embeddable into k-tree, inapproximability results.
- Triangulation heuristics: min-fill, min-size, randomization, etc.

Recap

- Triangulated graphs: if $|V| \geq 2$, always two simplicial nodes.
- Triangulated graph iff perfect elimination graph.
- All minimal triangulations of a graph can be created using elimination.
- k-trees, generalization of trees. Sometimes called hyper-tree. All min-separators are k-cliques. partial k-trees. Embedding into k-trees.
- Any triangulated graph G' can be embedded into k-tree where $k+1$ is the size of the largest clique of G'. Thus any graph can be embedded into a k-tree for large enough k.
- NP-complete: finding smallest k such that G is embeddable into k-tree, inapproximability results.
- Triangulation heuristics: min-fill, min-size, randomization, etc.
- MCS can identify a triangulated graph efficiently, produces a reverse elimination order.

MCS

- Can also produce an elimination order and triangulate the graphs (but not particularly good)
- will produce a perfect elimination order on triangulated graphs
- why called maximum cardinality "search"

Theorem 6.2.13

A graphical G is triangulated iff in the MCS algorithm, at each point when a vertex is marked, that vertex's previously marked neighbors form a complete subgraph of G.

Corollary 6.2.14

Every maximum cardinality search of a triangulated graph G corresponds to a reverse perfect eliminating order of G.

Multiple queries

- Let $\mathcal{C}$ be the set of all cliques in original graph. Often, we want to compute $p(x_C)$ for all $C \in \mathcal{C}$.

Multiple queries

- Let $\mathcal{C}$ be the set of all cliques in original graph. Often, we want to compute $p(x_C)$ for all $C \in \mathcal{C}$.
- Do not want to run separate elimination $|\mathcal{C}|$ many times.

Multiple queries

- Let $\mathcal{C}$ be the set of all cliques in original graph. Often, we want to compute $p(x_C)$ for all $C \in \mathcal{C}$.
- Do not want to run separate elimination $|\mathcal{C}|$ many times.
- Recall tree (i.e., 1-tree) case - messages for one query used for other queries. Message re-use/efficiency only grows with num. queries.

Multiple queries

- Let $\mathcal{C}$ be the set of all cliques in original graph. Often, we want to compute $p(x_C)$ for all $C \in \mathcal{C}$.
- Do not want to run separate elimination $|\mathcal{C}|$ many times.
- Recall tree (i.e., 1-tree) case - messages for one query used for other queries. Message re-use/efficiency only grows with num. queries. Can we do the same thing for arbitrary graphs?

Multiple queries

- Let $\mathcal{C}$ be the set of all cliques in original graph. Often, we want to compute $p(x_C)$ for all $C \in \mathcal{C}$.
- Do not want to run separate elimination $|\mathcal{C}|$ many times.
- Recall tree (i.e., 1-tree) case - messages for one query used for other queries. Message re-use/efficiency only grows with num. queries. Can we do the same thing for arbitrary graphs?
- Consider only the class of triangulated models since to do otherwise (for exact inference) is not necessary.

Multiple queries

- Let $\mathcal{C}$ be the set of all cliques in original graph. Often, we want to compute $p(x_C)$ for all $C \in \mathcal{C}$.
- Do not want to run separate elimination $|\mathcal{C}|$ many times.
- Recall tree (i.e., 1-tree) case - messages for one query used for other queries. Message re-use/efficiency only grows with num. queries. Can we do the same thing for arbitrary graphs?
- Consider only the class of triangulated models since to do otherwise (for exact inference) is not necessary.
- But is one triangulated model optimal for all queries?

Multiple queries

- A triangulated graph is a cover of G

Multiple queries

- A triangulated graph is a cover of G
- Any clique in G will still be a clique in a triangulation G': that is, given clique $c \in \mathcal{C}(G)$, there exists $c' \in \mathcal{C}(G')$ with $c \subseteq c'$.

Multiple queries

- A triangulated graph is a cover of G
- Any clique in G will still be a clique in a triangulation G': that is, given clique $c \in \mathcal{C}(G)$, there exists $c' \in \mathcal{C}(G')$ with $c \subseteq c'$.
- Given $p(x_{c'})$, can compute $p(x_c) = \sum_{x_{c' \setminus c}} p(x_{c'})$ at cost $O(r^{|c'|})$. Same cost triangulated graph.

Multiple queries

- A triangulated graph is a cover of G
- Any clique in G will still be a clique in a triangulation G': that is, given clique $c \in \mathcal{C}(G)$, there exists $c' \in \mathcal{C}(G')$ with $c \subseteq c'$.
- Given $p(x_{c'})$, can compute $p(x_c) = \sum_{x_{c' \setminus c}} p(x_{c'})$ at cost $O(r^{|c'|})$. Same cost triangulated graph.
- optimal k-tree embedding for G is one that minimizes the maximum clique for any triangulation of G, so if we have found this embedding, this will be optimal for any original-graph clique marginal.

Multiple queries

- A triangulated graph is a cover of G
- Any clique in G will still be a clique in a triangulation G': that is, given clique $c \in \mathcal{C}(G)$, there exists $c' \in \mathcal{C}(G')$ with $c \subseteq c'$.
- Given $p(x_{c'})$, can compute $p(x_c) = \sum_{x_{c' \setminus c}} p(x_{c'})$ at cost $O(r^{|c'|})$. Same cost triangulated graph.
- optimal k-tree embedding for G is one that minimizes the maximum clique for any triangulation of G, so if we have found this embedding, this will be optimal for any original-graph clique marginal.
- Even if we found a "good" elimination order (one that produces a maxclique of reasonable size), this order can be shared for other clique queries.

Non-clique queries

- Recall: 1-tree case, if we want a marginal over a non-sub-tree, we might be in trouble.

Non-clique queries

- Recall: 1-tree case, if we want a marginal over a non-sub-tree, we might be in trouble.
- Similarly, if we desire non-clique queries for general graph, then computation can get worse. Computing $p(x_L)$ for arbitrary L could turn x_L into a clique in the worst case (Rose's theorem).

Non-clique queries

- Recall: 1-tree case, if we want a marginal over a non-sub-tree, we might be in trouble.
- Similarly, if we desire non-clique queries for general graph, then computation can get worse. Computing $p(x_L)$ for arbitrary L could turn x_L into a clique in the worst case (Rose's theorem).
- If x_L is not clique in G', then we can view G' as not being "valid" for the query $p(x_L)$.

Non-clique queries

- Recall: 1-tree case, if we want a marginal over a non-sub-tree, we might be in trouble.
- Similarly, if we desire non-clique queries for general graph, then computation can get worse. Computing $p(x_L)$ for arbitrary L could turn x_L into a clique in the worst case (Rose's theorem).
- If x_L is not clique in G', then we can view G' as not being "valid" for the query $p(x_L)$.
- In such case, need to re-triangulate, starting with a graph where x_L is made complete.

Computing all clique queries efficiently via elimination

- Remarkably, in the case of clique queries, we can actually re-use the elimination order.

Computing all clique queries efficiently via elimination

- Remarkably, in the case of clique queries, we can actually re-use the elimination order.
- We want to share more than just the elimination order.

Computing all clique queries efficiently via elimination

- Remarkably, in the case of clique queries, we can actually re-use the elimination order.
- We want to share more than just the elimination order.
- goal: in non-tree graphs, re-use work of computing marginals for the sake of getting multiple marginals.

Computing all clique queries efficiently via elimination

- Remarkably, in the case of clique queries, we can actually re-use the elimination order.
- We want to share more than just the elimination order.
- goal: in non-tree graphs, re-use work of computing marginals for the sake of getting multiple marginals.
- We'll see an amazing fact: if we find the optimal elimination order for 1 clique query, it is optimal for all clique queries!! ☺

Decomposition of G

Definition 6.3.1 (Decomposition of G)

A *decomposition* of a graph $G = (V, E)$ (if it exists) is a partition (A, B, C) of V such that:

- C separates A from B in G.
- C is a clique.

if A and B are both non-empty, then the decomposition is called *proper*.

If G has a decomposition, what dies this mean for the family $\mathcal{F}(G, \mathcal{M}^{(\mathsf{f})})$? Since C separates A from B, this means that $X_A \perp\!\!\!\perp X_B | X_C$ for any $p \in \mathcal{F}(G, \mathcal{M}^{(\mathsf{f})})$, which moreover means we can write the joint distribution in a particular form.

$$p(x) = p(x_A, x_B, x_C) = \frac{p(x_A, x_C)p(x_B, x_C)}{p(x_C)} \tag{6.1}$$

Decomposable models

Definition 6.3.2

A graph $G = (V, E)$ is decomposable if either: 1) G is a clique, or 2) G possesses a proper decomposition (A, B, C) s.t. both subgraphs $G[A \cup C]$ and $G[B \cup C]$ are decomposable.

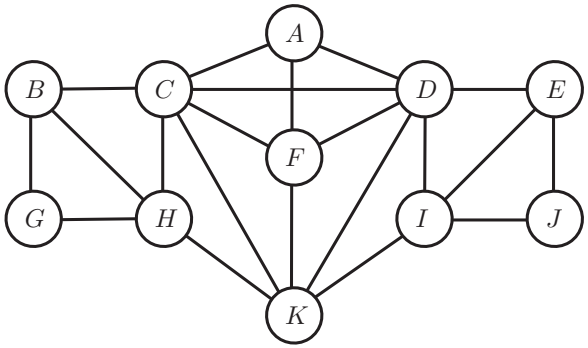
Decomposable models

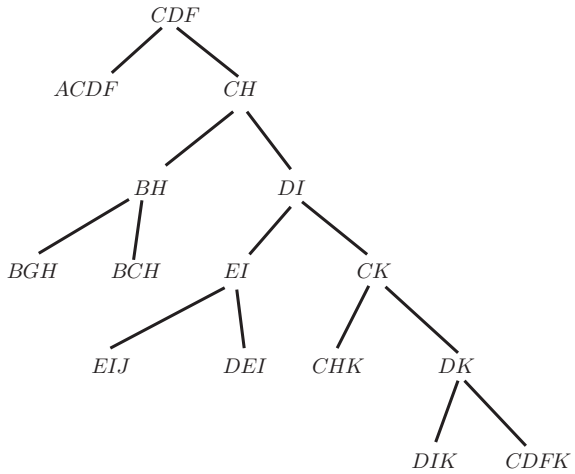
Definition 6.3.2

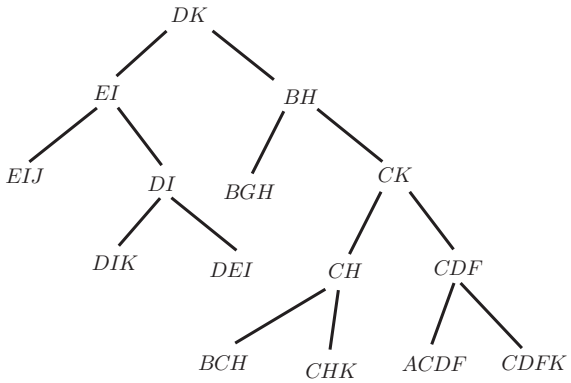
A graph $G = (V, E)$ is decomposable if either: 1) G is a clique, or 2) G possesses a proper decomposition (A, B, C) s.t. both subgraphs $G[A \cup C]$ and $G[B \cup C]$ are decomposable.

- Note that the separator is contained within the subgraphs: i.e., $G[A \cup C]$ rather than, say, $G[A]$.

Decomposable models

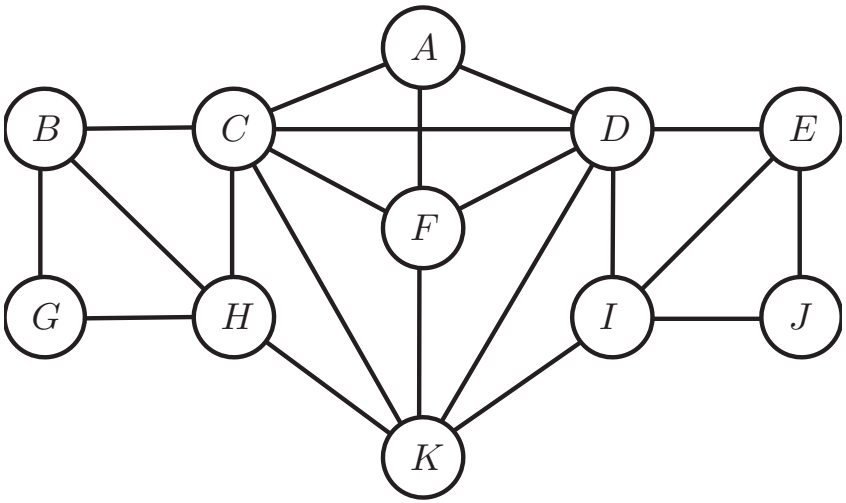


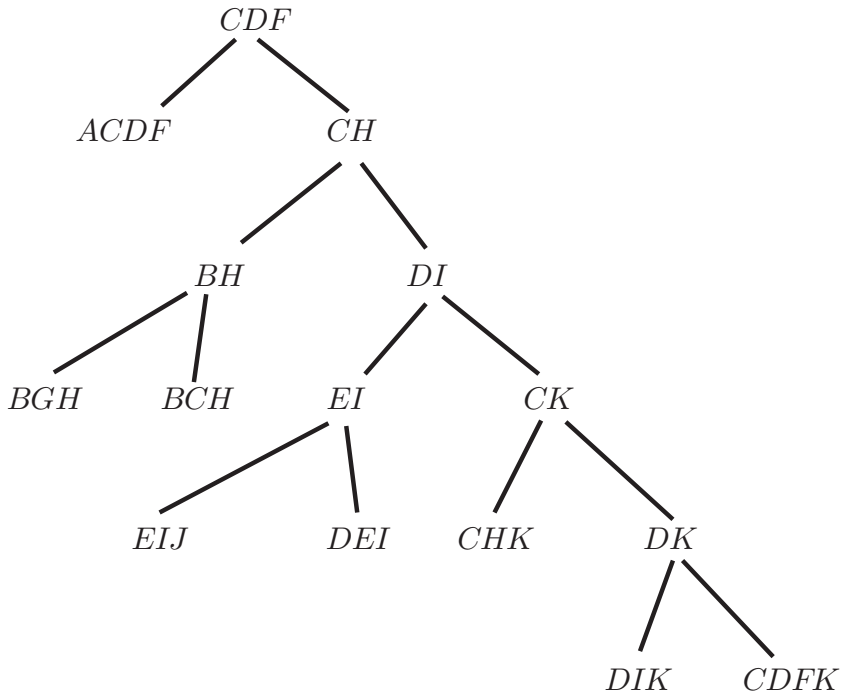




- Graph and two decompositions of this graph.
- as we recurse down, if at any point decomposition is not found, graph is not decomposable.

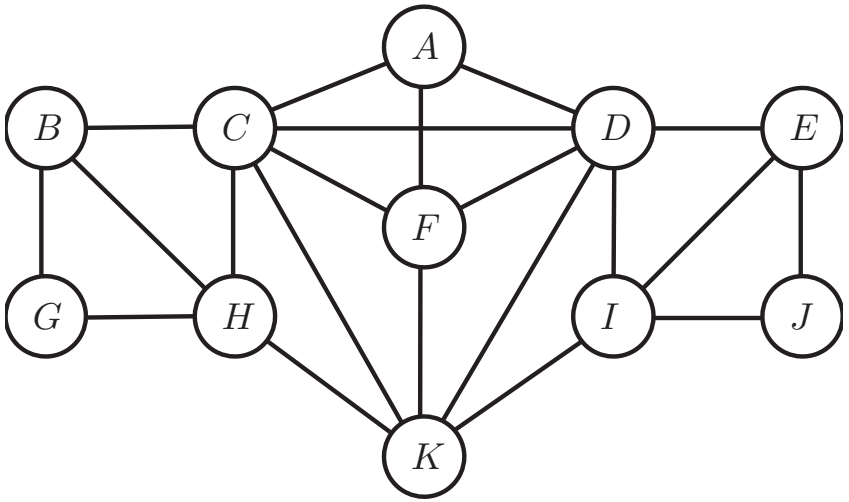
Decomposable models

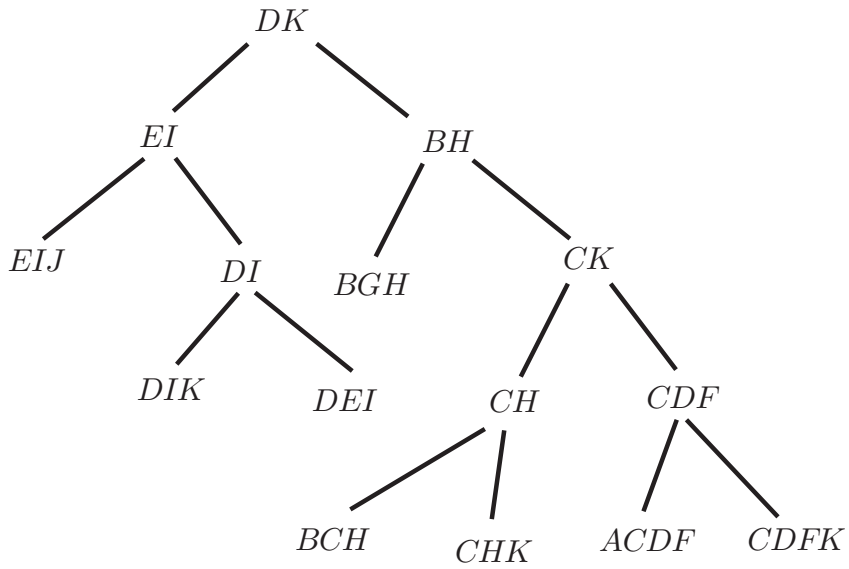




- Graph and two decompositions of this graph.
- as we recurse down, if at any point decomposition is not found, graph is not decomposable.

Decomposable models





- Graph and two decompositions of this graph.
- as we recurse down, if at any point decomposition is not found, graph is not decomposable.

Decomposition of G and Decomposable graphs

Repeat of both definitions, but on one page.

Definition 6.3.3 (Decomposition of G)

A *decomposition* of a graph $G = (V, E)$ (if it exists) is a partition (A, B, C) of V such that:

- C separates A from B in G.
- C is a clique.

if A and B are both non-empty, then the decomposition is called *proper*.

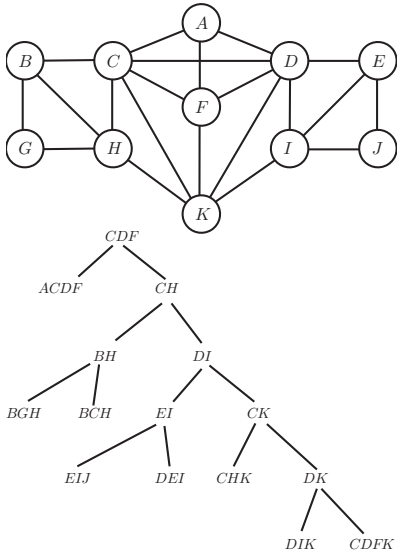
Definition 6.3.4

A graph $G = (V, E)$ is decomposable if either: 1) G is a clique, or 2) G possesses a **proper** decomposition (A, B, C) s.t. both subgraphs $G[A \cup C]$ and $G[B \cup C]$ are decomposable.

Note part 2. It says *possesses*. Bottom of tree might affect top.

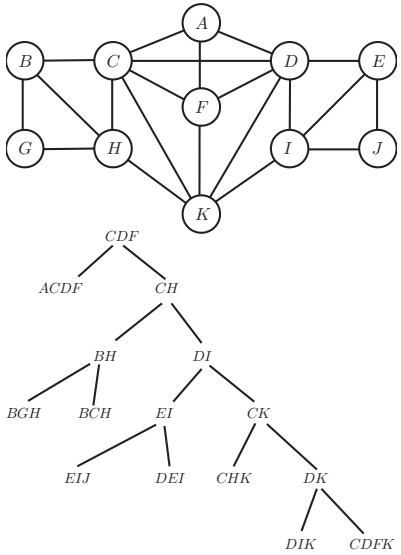
Decomposable models

- Internal nodes in tree are complete graphs that are also separators.

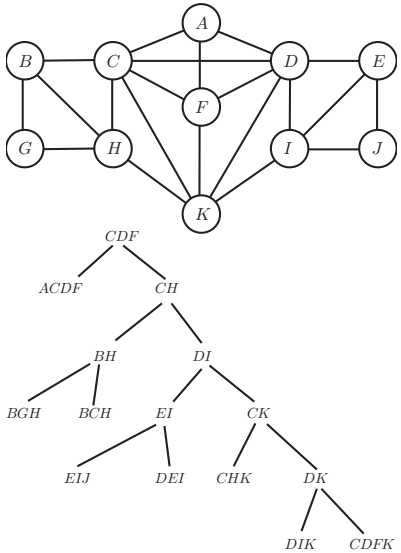


Decomposable models

- Internal nodes in tree are complete graphs that are also separators.
- When G is decomposable, what are implications for a $p \in \mathcal{F}(G, \mathcal{M}^{(\mathsf{f})})$?



Decomposable models



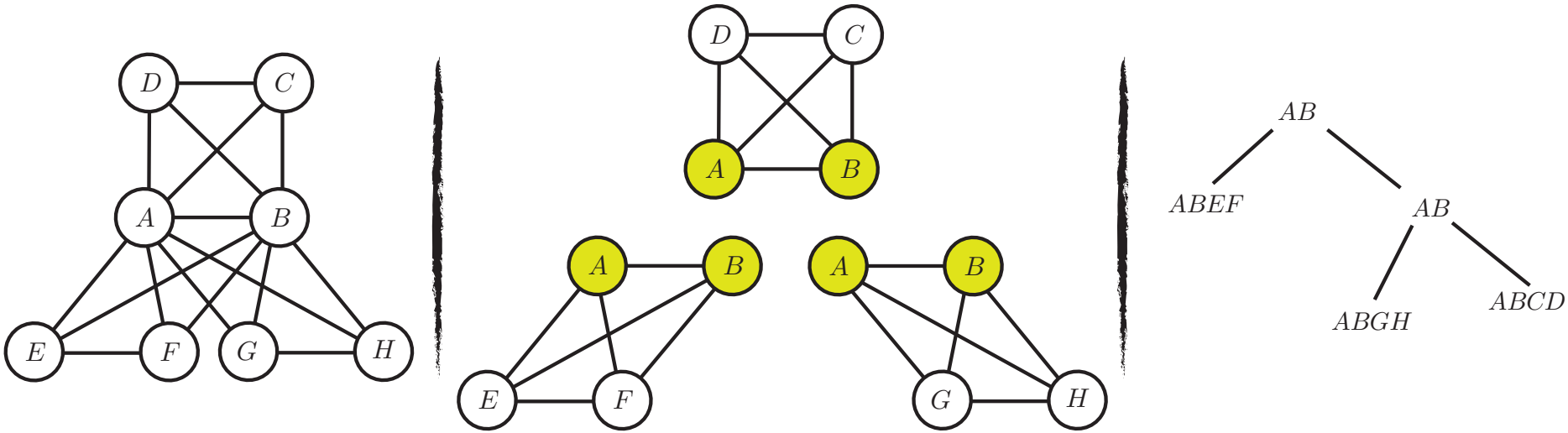
- Internal nodes in tree are complete graphs that are also separators.
- When G is decomposable, what are implications for a $p \in \mathcal{F}(G, \mathcal{M}^{(\mathsf{f})})$?

$$
\begin{aligned}
& p(A,B,C,D,E,F,G,H,I,J,K) \\
&= \frac{p(A,C,D,F)p(B,C,D,E,F,G,H,I,J,K)}{p(C,D,F)} \\
&= \frac{p(A,C,D,F)}{p(C,D,F)} \left(\frac{p(B,C,G,H)p(C,D,E,F,H,I,J,K)}{p(C,H)} \right) \\
&= \ldots
\end{aligned}
$$

$$
= \frac{p(A,C,D,F)p(B,G,H)p(C,B,H)p(I,E,J)p(E,I,D)p(C,K,H)p(D,K,I)p(D,K,F,C)}{p(C,D,F)p(C,H)p(B,H)p(D,I)p(E,I)p(C,K)p(D,K)}
$$

Shattering of a graph

- S is a separator, so that $G[V \setminus S]$ consists of 2 or more **connected components**.
- We say that S *shatters* the graph G into those components, and let $d(S)$ be the number of connected components that S shatters G into. $d(S)$ is the shattering coefficient of G.
- Example: below, $d(\{A, B\}) = 3$



Decomposable models

- When $d(S) > 2$, separator marginal use more than once in the denominator
- The general form of the factorization becomes:

$$p(x) = \frac{\prod_{C \in \mathcal{C}(G)} p(x_C)}{\prod_{S \in \mathcal{S}(G)} p(x_S)^{d(S)-1}} \tag{6.2}$$

- Any decomposable model can be written this way
- 4-cycle is not decomposable. Two independence properties that can't be used simultaneously.

$$p(x_1, x_2, x_3, x_4) = \frac{p(x_1, x_2, x_4)p(x_1, x_3, x_4)}{p(x_1, x_4)} = \frac{p(x_1, x_2, x_3)p(x_2, x_3, x_4)}{p(x_2, x_3)} \tag{6.3}$$

Decomposable models

Proposition 6.3.5

All of the maxcliques in a graph lie on the leaf nodes of the binary decomposition tree

Proof.

For a decomposable model, the base case (leaf node) is a clique, otherwise it would not be decomposable. If a leaf was not a maxclique (and only a clique), then that means it is contained in a maxclique, and got split by a separator corresponding to that leaf's parent, but this is impossible since a maxcliques have no separator. □

Proposition 6.3.6

The (nec. unique) set of all minimal separators of graph are included in the non-leaf nodes of the binary decomposition tree. $d(S) - 1$ *is the number of times the minimal separator* S *appears as a given non-leaf node.*

A bit of notation

- If C is separator, C shatters G into $d(C)$ connected components
- $G[V \setminus C]$ is the union of these components (not including C)
- Let $\{G_1, G_2, \ldots, G_\ell\}$ be (disjoint) connected components of $G[V \setminus C]$, so $G_1 \cup G_2 \cup \cdots \cup G_\ell = G[V \setminus C]$
- Given $a \in V(G_i)$ for some i, then $G[V \setminus C](a) = G_i$.

Triangulated vs. decomposable

Theorem 6.3.7

A given graph $G = (V, E)$ is triangulated iff it is decomposable.

Proof.

First, recall from Lemma 4.5.6 that a graph is triangulated iff it is decomposable. To prove the current theorem, we will first show (by induction) that decomposability implies that the graph is triangulated). Next, for the converse, we'll show (also by induction on $n = |V|$) that every minimal separator complete in G implies decomposable.

Triangulated vs. decomposable

Proof of Theorem 6.3.7.

First, assume G is decomposable. If G is complete then it is triangulated. If it is not complete then there exists a proper decomposition (A, B, C) into decomposable subgraphs $G[A \cup C]$ and $G[B \cup C]$ both of which have fewer vertices, meaning $|A \cup C| < |V|$ and $|B \cup C| < |V|$. By the induction hypothesis, both $G[A \cup C]$ and $G[B \cup C]$ are chordal. Any potential chordless cycle, therefore, can't be contained in one of the sub-components, so if it exist in G must intersect both A and B. Since C separates A from B, the purported chordless cycle would intersect C twice, but C is complete the cycle has a chord. The first part of the theorem is proven.

Triangulated vs. decomposable

. . . proof of Theorem 6.3.7 cont.

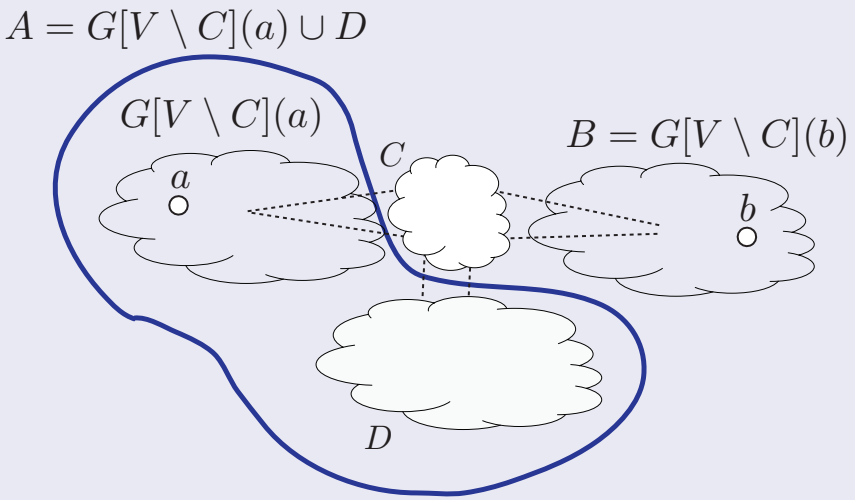
For the converse, assume all minimum (a, b) separators are complete in G, and assume (induction hypothesis) that all min. (a, b) separators are complete implies decomposable for smaller graphs.

Triangulated vs. decomposable

. . . proof of Theorem 6.3.7 cont.

For the converse, assume all minimum (a, b) separators are complete in G, and assume (induction hypothesis) that all min. (a, b) separators are complete implies decomposable for smaller graphs.

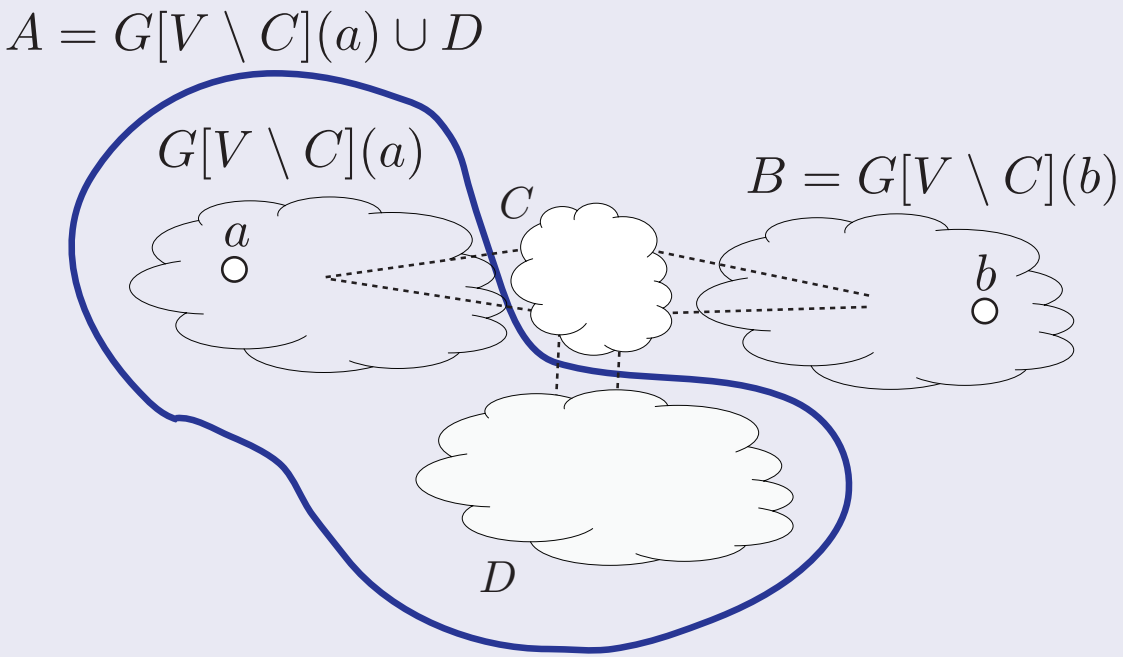
If G is complete then it is decomposable. Otherwise, there exists two non-adjacent vertices $a, b \in V$ in G with a necessarily complete minimal separator C forming a partition $G[V \setminus C](a)$, $G[V \setminus C](b)$, and all of the remaining components of $G[V \setminus C]$. We merge the connected components together to form only two components as follows: let $A = G[V \setminus C](a) \cup D$ and $B = G[V \setminus C](b)$.



Triangulated vs. decomposable

. . . proof of Theorem 6.3.7 cont.

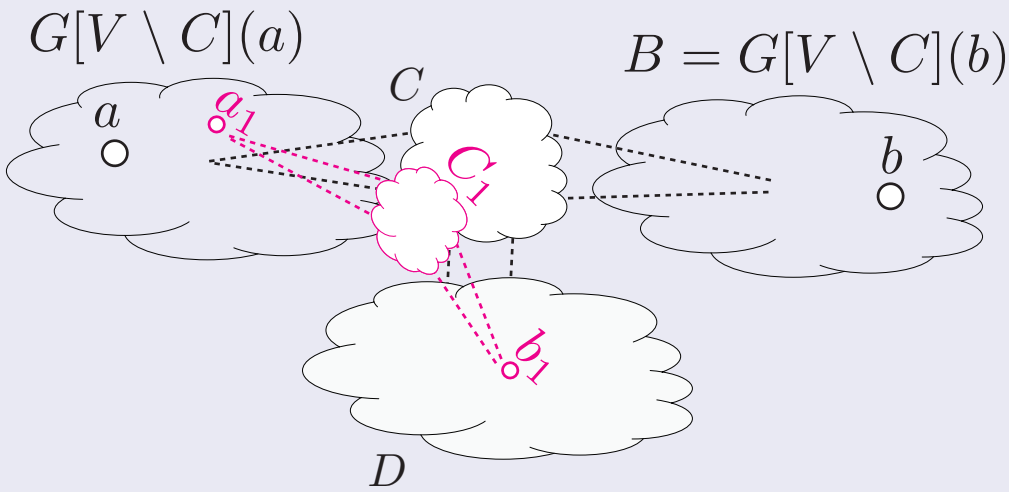
Since C is complete, we see that (A, B, C) form a decomposition of G, but we still need that $G[A \cup C]$ and $G[B \cup C]$ to be decomposable (see figure). Lets consider the decomposability if A first.



Triangulated vs. decomposable

. . . proof of Theorem 6.3.7 cont.

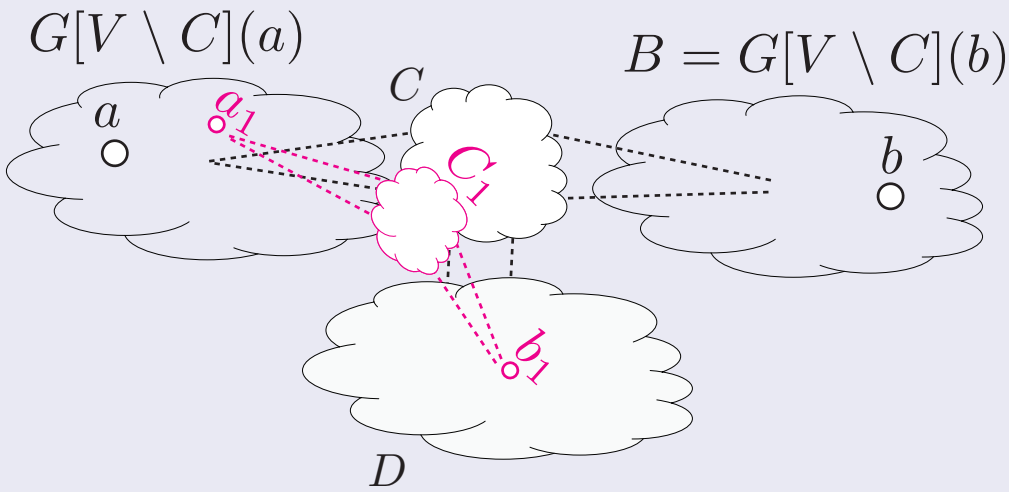
Considering A, let C_1 be a minimal (a_1, b_1) separator in $G[A \cup C]$ (see right). Is C_1 also minimal (and thus complete) in G as well?



Triangulated vs. decomposable

. . . proof of Theorem 6.3.7 cont.

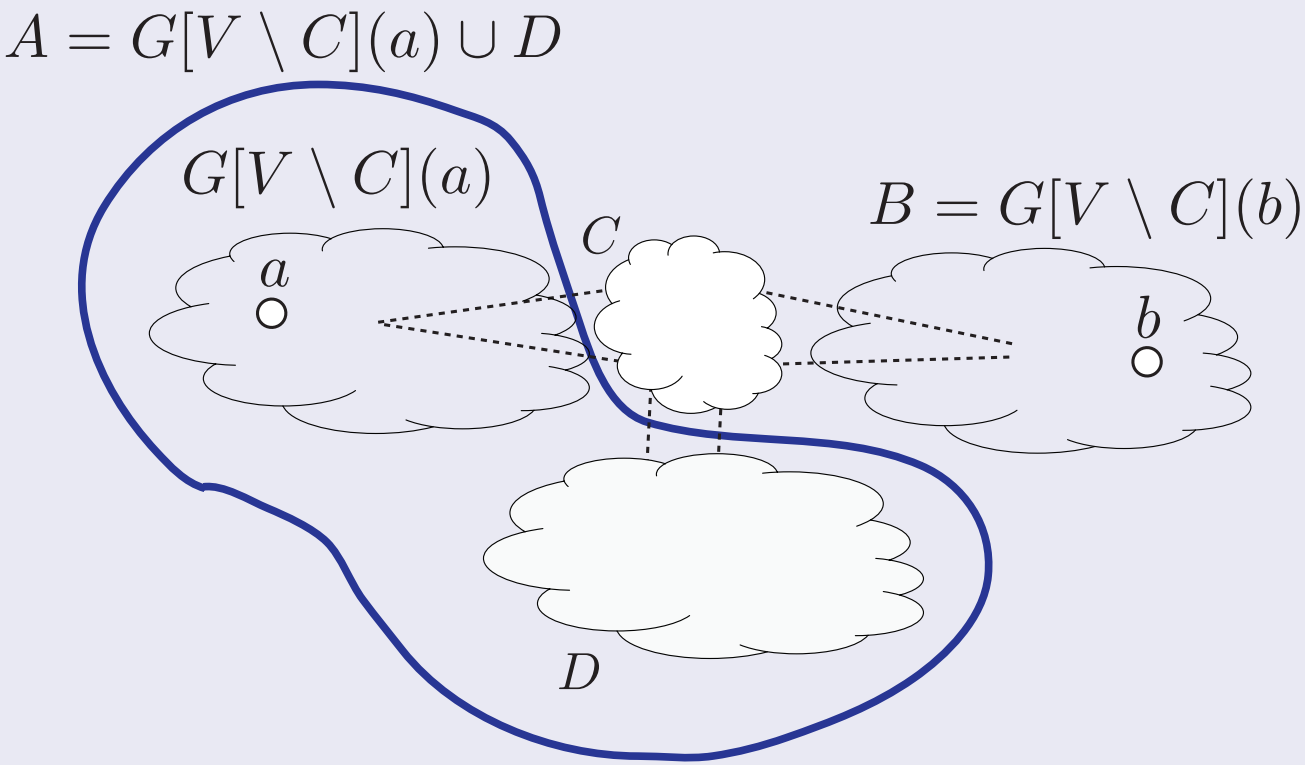
Considering A, let C_1 be a minimal (a_1, b_1) separator in $G[A \cup C]$ (see right). Is C_1 also minimal (and thus complete) in G as well?



Yes, C_1 is also a minimal (a_1, b_1) separator in G since, once we add B back to $G[A \cup C]$ to get G, there are no new paths from a_1 to b_1 circumventing C_1. This is because any such path would involve nodes in B (the only new nodes) which, to reach B and return, requires going through C (which is complete) twice. Such a path cannot bypass C_1 since if it did, a shorter path not involving B would bypass C_1, contradicting C_1 being a separator. Therefore, C_1 is complete in G, and hence complete in $G[A \cup C]$, and an inductive argument says that $G[A \cup C]$ is decomposable.

Triangulated vs. decomposable

. . . proof of Theorem 6.3.7 cont.



The same argument that held for A also holds for B in the graph $G[B \cup C]$. Therefore, G is decomposable.

Tree decomposition

Definition 6.3.8 (tree decomposition)

Given a graph $G = (V, E)$, a tree-decomposition of a graph is a pair $(\{C_i : i \in I\}, T)$ where $T = (I, F)$ is a tree with node index set I, edge set F, and $\{C_i\}_i$ (one for each $i \in I$) is a collection of subsets of $V(G)$ such that:

1. $\cup_{i \in I} C_i = V$
2. for any $(u, v) \in E(G)$, there exists $i \in I$ with $u, v \in C_i$
3. for any $v \in V$, the set $\{i \in I : v \in C_i\}$ forms a connected subtree of T

Tree decomposition is also hard

- Definition: The tree-width of the tree-decomposition is the size of the largest C_i minus one (i.e., $\max_{i \in I} |C_i| - 1$.

Tree decomposition is also hard

- Definition: The tree-width of the tree-decomposition is the size of the largest C_i minus one (i.e., $\max_{i \in I} |C_i| - 1$.

Theorem 6.3.9

Given graph $G = (V, E)$, finding the tree decomposition $T = (I, F)$ of G that minimizes the tree width ($\max_{i \in I} |C_i| - 1$) is an NP-complete optimization problem.

Tree decomposition is also hard

- Definition: The tree-width of the tree-decomposition is the size of the largest C_i minus one (i.e., $\max_{i \in I} |C_i| - 1$.

Theorem 6.3.9

Given graph $G = (V, E)$, finding the tree decomposition $T = (I, F)$ of G that minimizes the tree width ($\max_{i \in I} |C_i| - 1$) is an NP-complete optimization problem.

- How does this relate to our problem though?

$\longrightarrow$ trees

- All roads lead to trees, namely junction trees.

- Next set of slides will make the transformation mathematically precise.

Cluster graphs

Definition 6.4.1 (Cluster graph)

Consider forming a new graph based on G where the new graph has nodes that correspond to clusters in the original G, and has edges existing between two (cluster) nodes only when the corresponding clusters have a non-zero intersection. That is, let $\mathcal{C}(G) = \{C_1, C_2, \ldots, C_{|I|}\} =$ be a set of $|I|$ clusters of nodes $V(G)$, where $C_i \subseteq V(G), i \in I$. Consider a new graph $\mathcal{J} = (I, \mathcal{E})$ where each node in $\mathcal{J}$ corresponds to a set of nodes in G, and where edge $(i, j) \in \mathcal{E}$ if $C_i \cap C_j \neq \emptyset$. We will also use $S_{ij} = C_i \cap C_j$ as notation.

Cluster graphs

Definition 6.4.1 (Cluster graph)

Consider forming a new graph based on G where the new graph has nodes that correspond to clusters in the original G, and has edges existing between two (cluster) nodes only when the corresponding clusters have a non-zero intersection. That is, let $\mathcal{C}(G) = \{C_1, C_2, \ldots, C_{|I|}\} =$ be a set of $|I|$ clusters of nodes $V(G)$, where $C_i \subseteq V(G), i \in I$. Consider a new graph $\mathcal{J} = (I, \mathcal{E})$ where each node in $\mathcal{J}$ corresponds to a set of nodes in G, and where edge $(i, j) \in \mathcal{E}$ if $C_i \cap C_j \neq \emptyset$. We will also use $S_{ij} = C_i \cap C_j$ as notation.

So two cluster nodes have an edge between them iff there is non-zero intersection between the nodes.

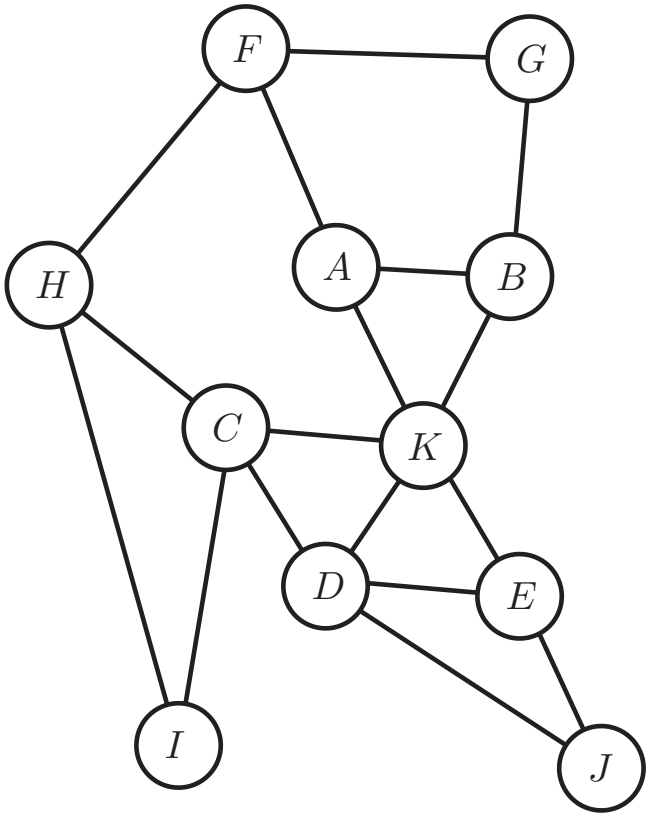
Cluster Trees

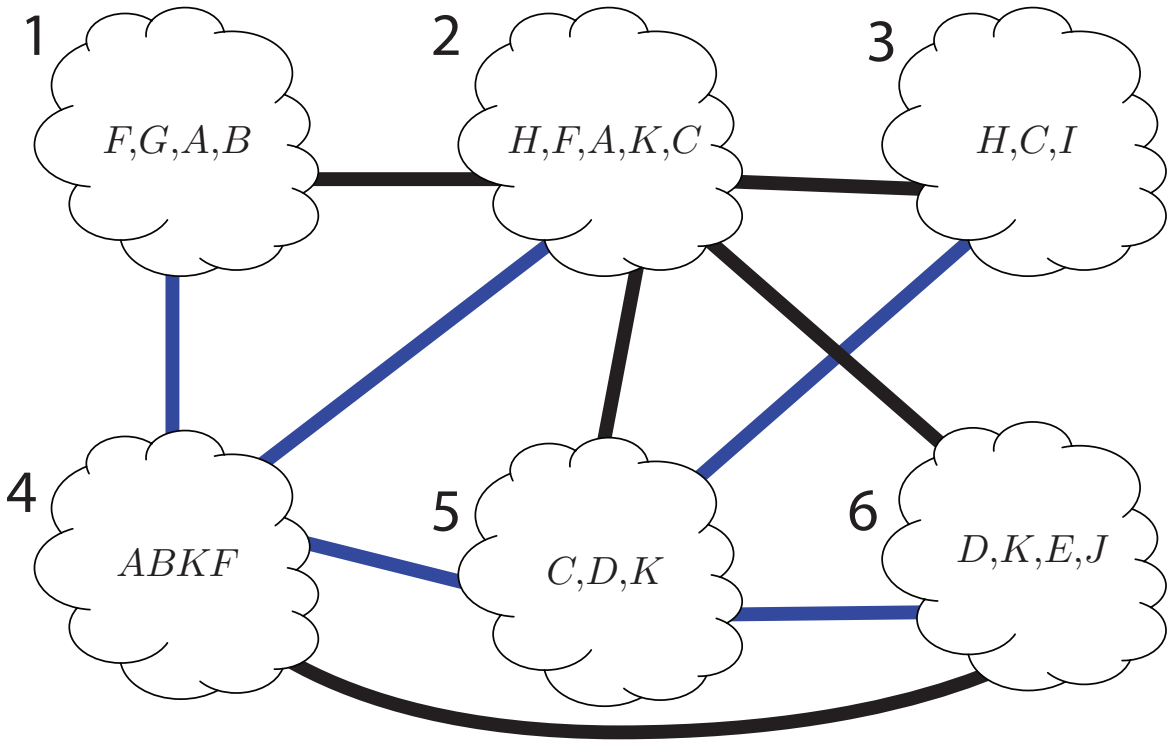
If the graph is a tree, then we have what is called a cluster tree.

Definition 6.4.2 (Cluster Tree)

Let $\mathcal{C} = \{C_1, C_2, \ldots, C_{|I|}\}$ be a set of node clusters of graph $G = (V, E)$. A cluster tree is a tree $\mathcal{T} = (I, \mathcal{E}_T)$ with vertices corresponding to clusters in $\mathcal{C}$ and edges corresponding to pairs of clusters $C_1, C_2 \in \mathcal{C}$. We can label each vertex in $i \in I$ by the set of graph nodes in the corresponding cluster in G, and we label each edge $(i, j) \in \mathcal{E}_T$ by the cluster intersection, i.e., $S_{ij} = C_i \cap C_j$.

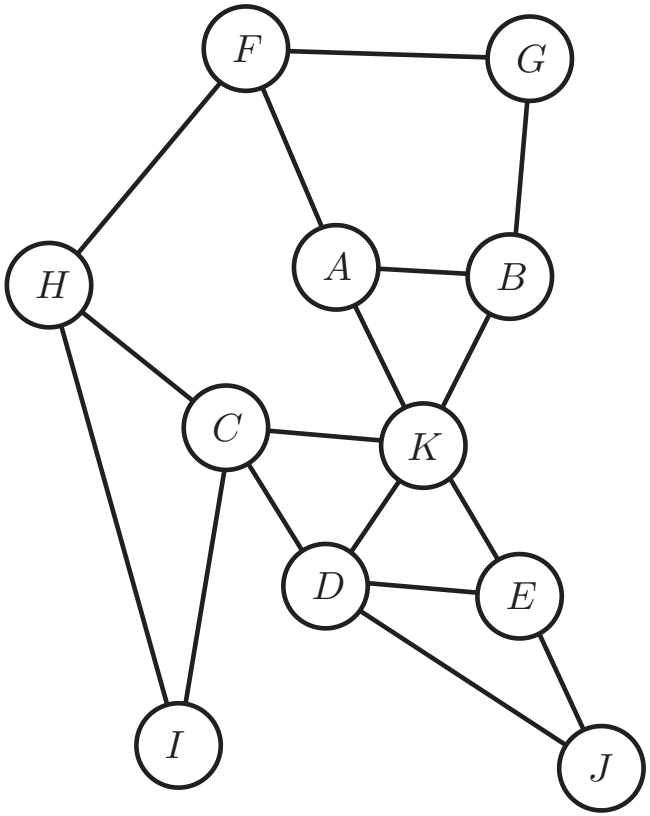
Cluster Graphs/Trees

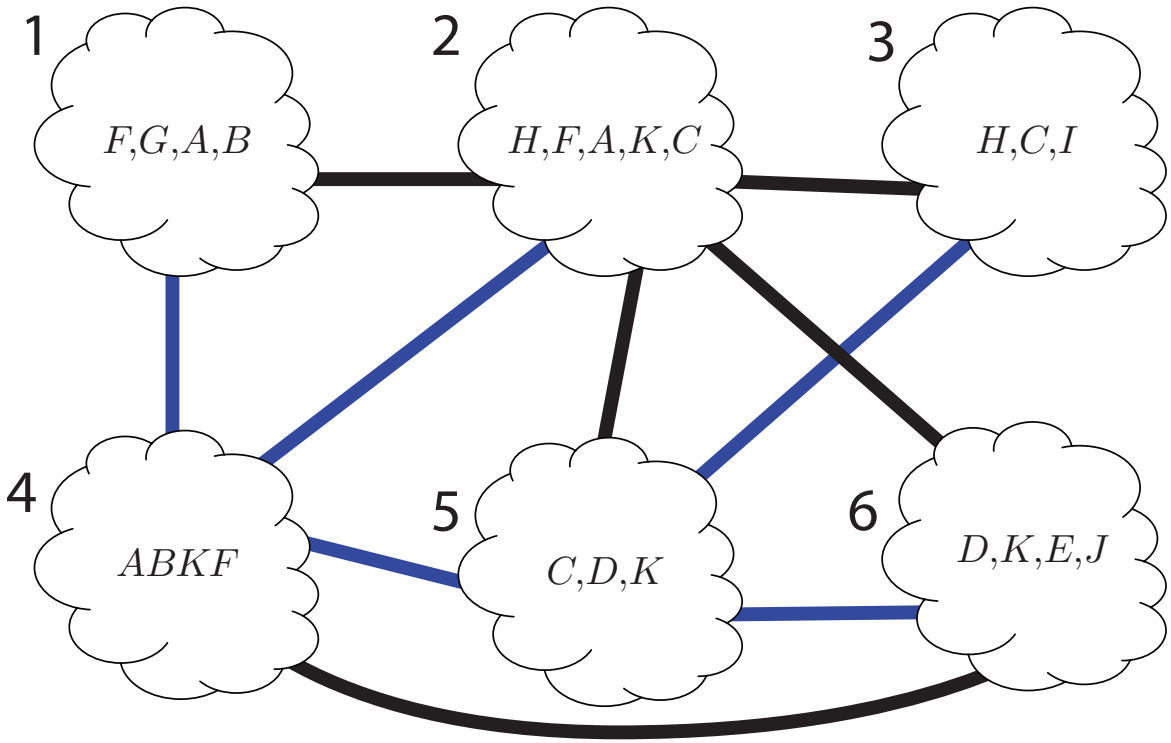




Left: a graph. Right: A cluster graph with $|I| = 6$ clusters, where $C_1 = \{F, G, A, B\}, C_2 = \{H, F, A, K, C\}, \ldots$. There is an edge $(1, 2)$ since $C_1 \cap C_2 = \{F, A\} \neq \emptyset$.

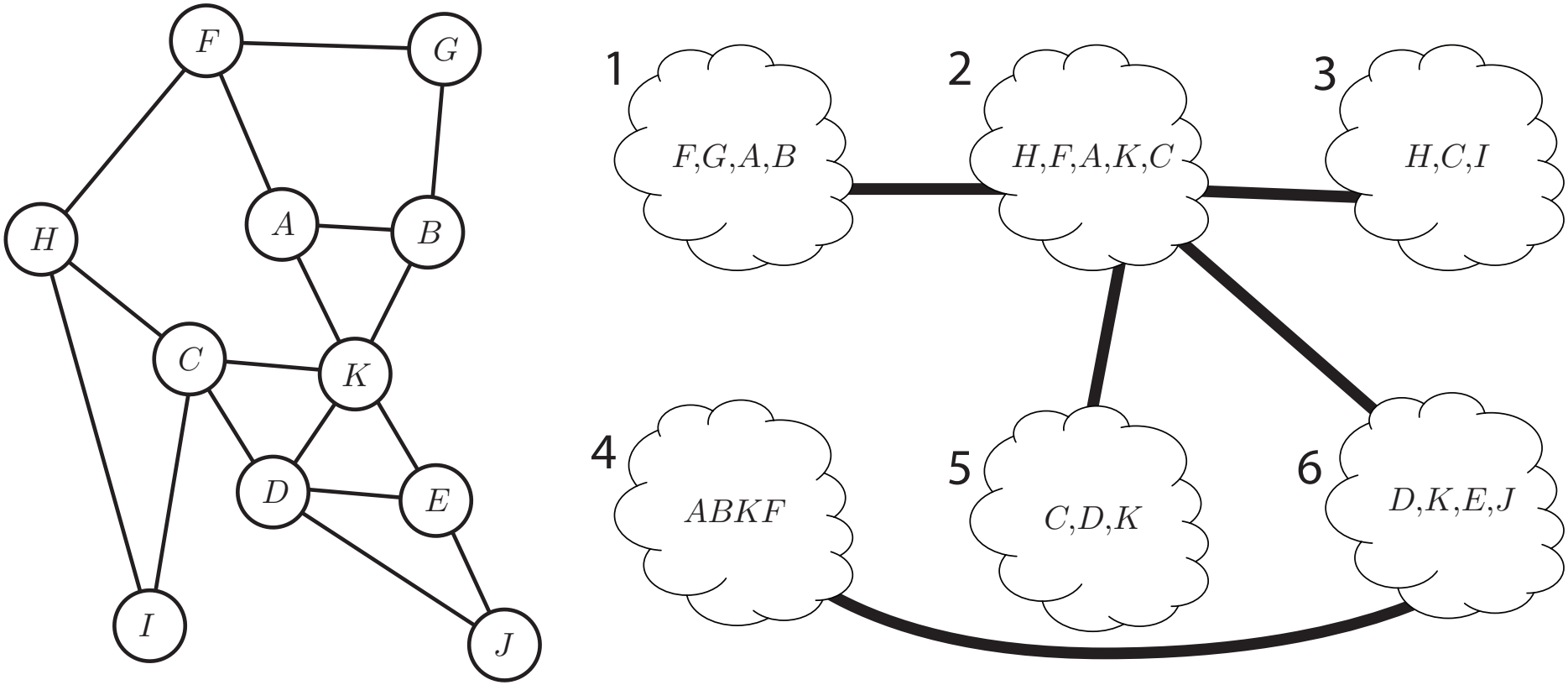
Cluster Graphs/Trees





Left: a graph. Right: A cluster graph with $|I| = 6$ clusters, where $C_1 = \{F, G, A, B\}, C_2 = \{H, F, A, K, C\}, \ldots$. There is an edge $(1, 2)$ since $C_1 \cap C_2 = \{F, A\} \neq \emptyset$. If we remove all but the blue edges, then we get a cluster tree.

Cluster Graphs/Trees



Left: a graph. Right: A cluster graph with $|I| = 6$ clusters, where $C_1 = \{F, G, A, B\}, C_2 = \{H, F, A, K, C\}, \ldots$. There is an edge $(1, 2)$ since $C_1 \cap C_2 = \{F, A\} \neq \emptyset$.

Cluster Intersection Property (c.i.p.)

- Important: Cluster graphs and cluster trees are based only on a set of clusters of nodes of $G = (V, E)$. We haven't, based on these definitions, yet used any of the original graph (o.g.) edges of G.
- Edges in a cluster graph and cluster tree are not o.g. edges. Instead, they are based on if two clusters have non-empty intersection.
- We want to talk about cluster trees that have certain properties. A cluster graph might or might not have such properties.

Cluster Intersection Property (c.i.p.)

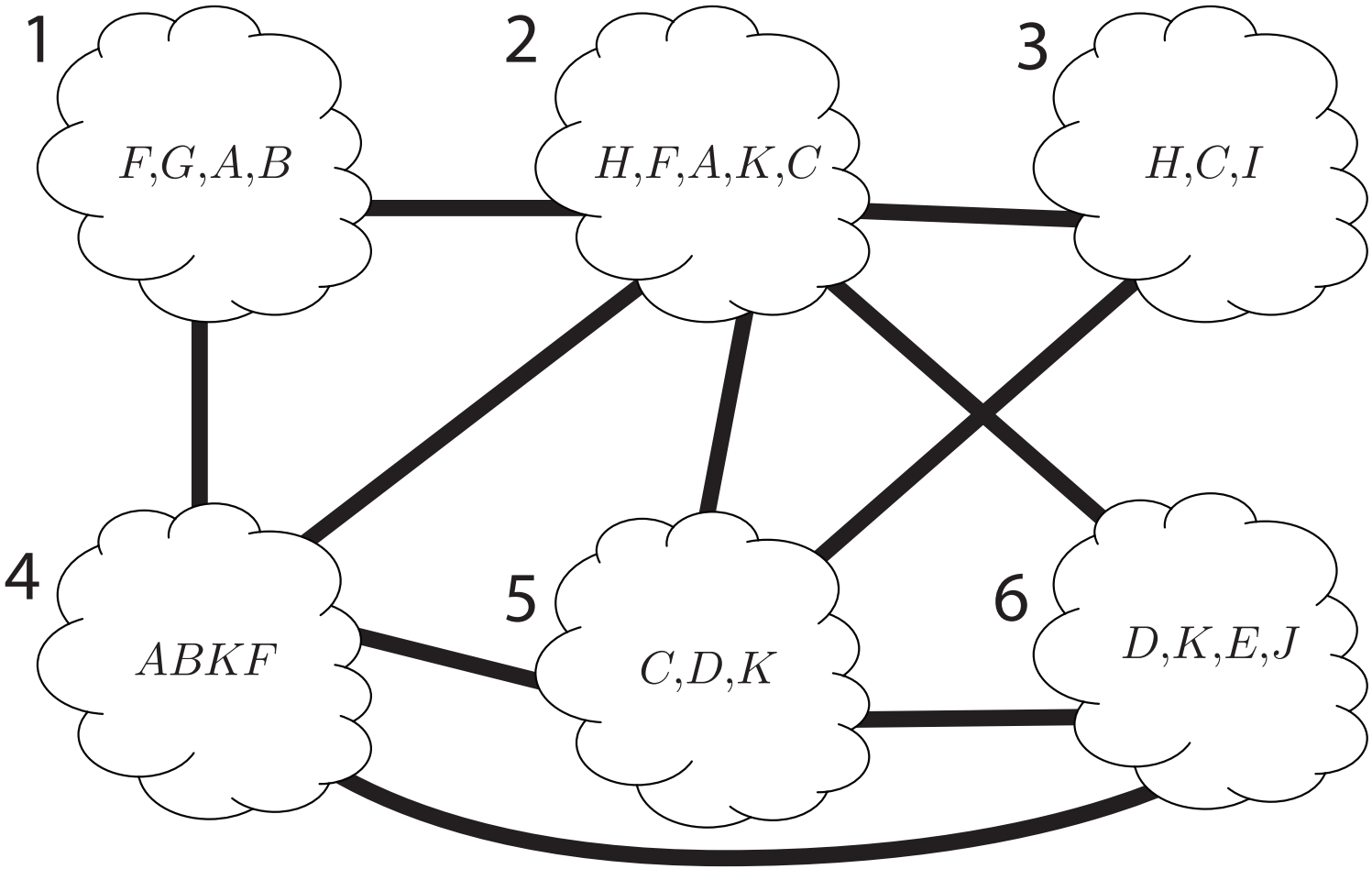
Definition 6.4.3 (Cluster Intersection Property)

We are given a cluster tree $\mathcal{T} = (I, \mathcal{E}_T)$, and let C_1, C_2 be any two clusters in the tree. Then the cluster intersection property states that $C_1 \cap C_2 \subseteq C_i$ for all C_i on the (by definition, necessarily) unique path between C_1 and C_2 in the tree $\mathcal{T}$.

- A given cluster tree might or might not have that property.
- Example on the next few slides.

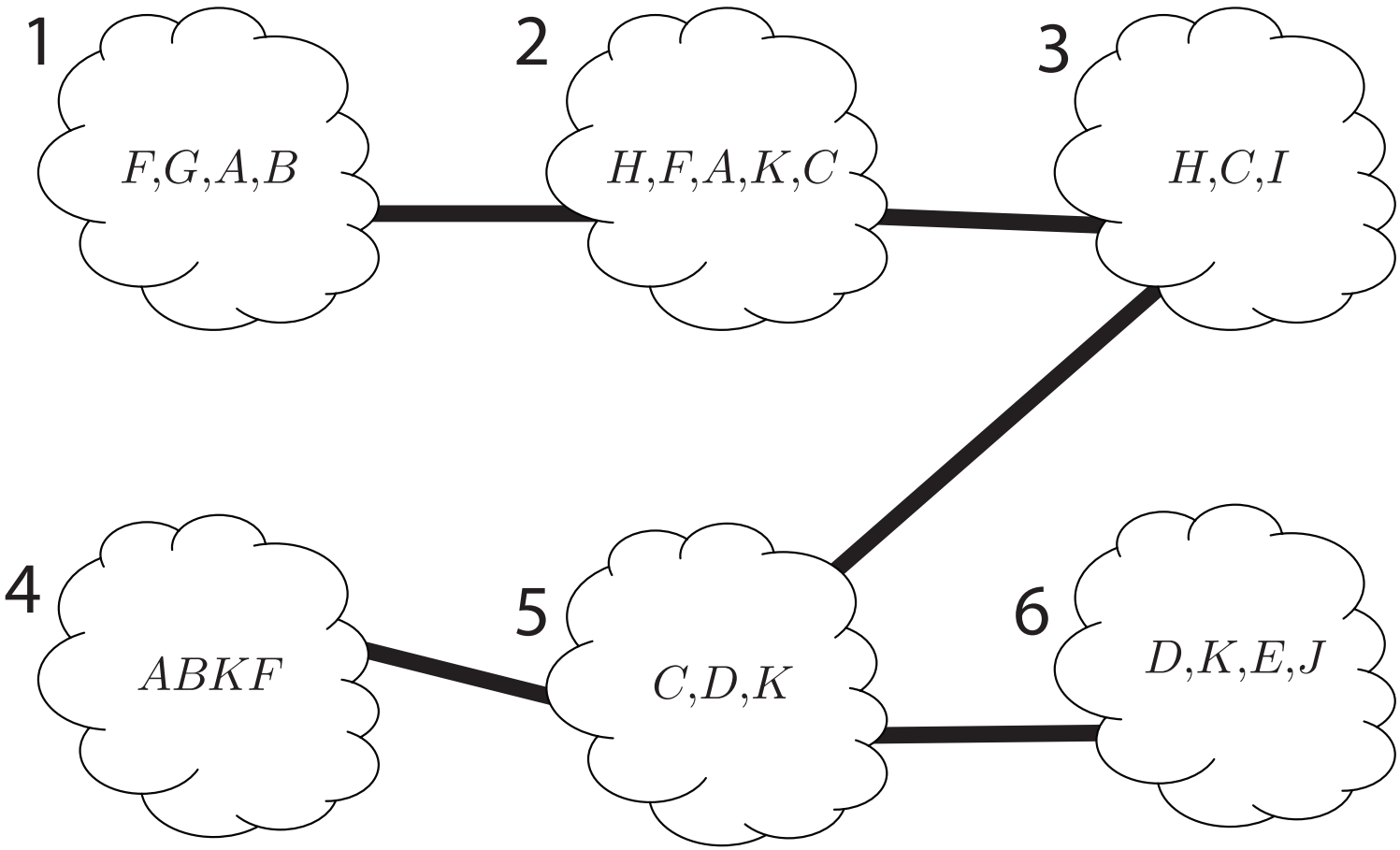
Examples

Cluster Graph



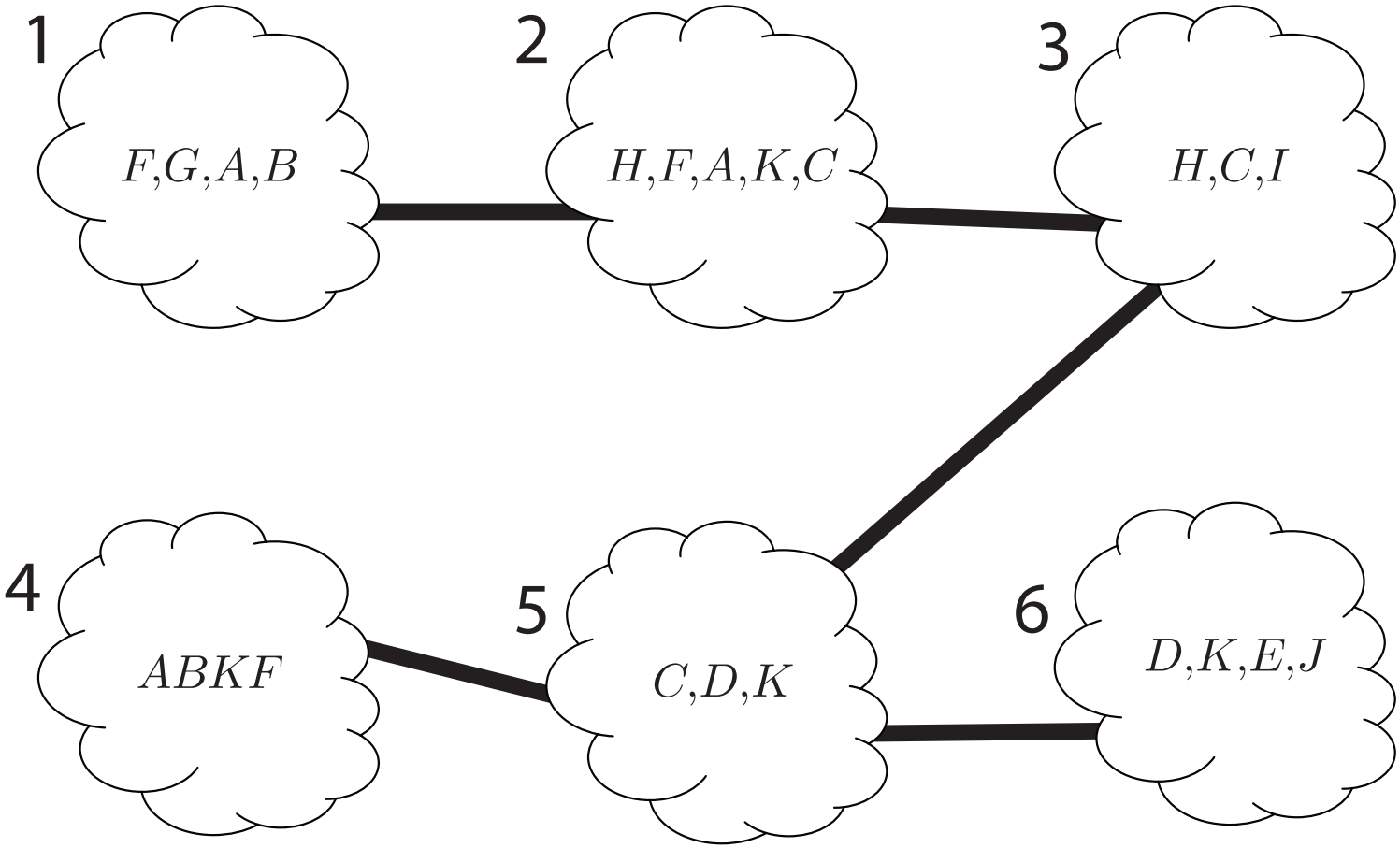
Examples

Cluster Tree



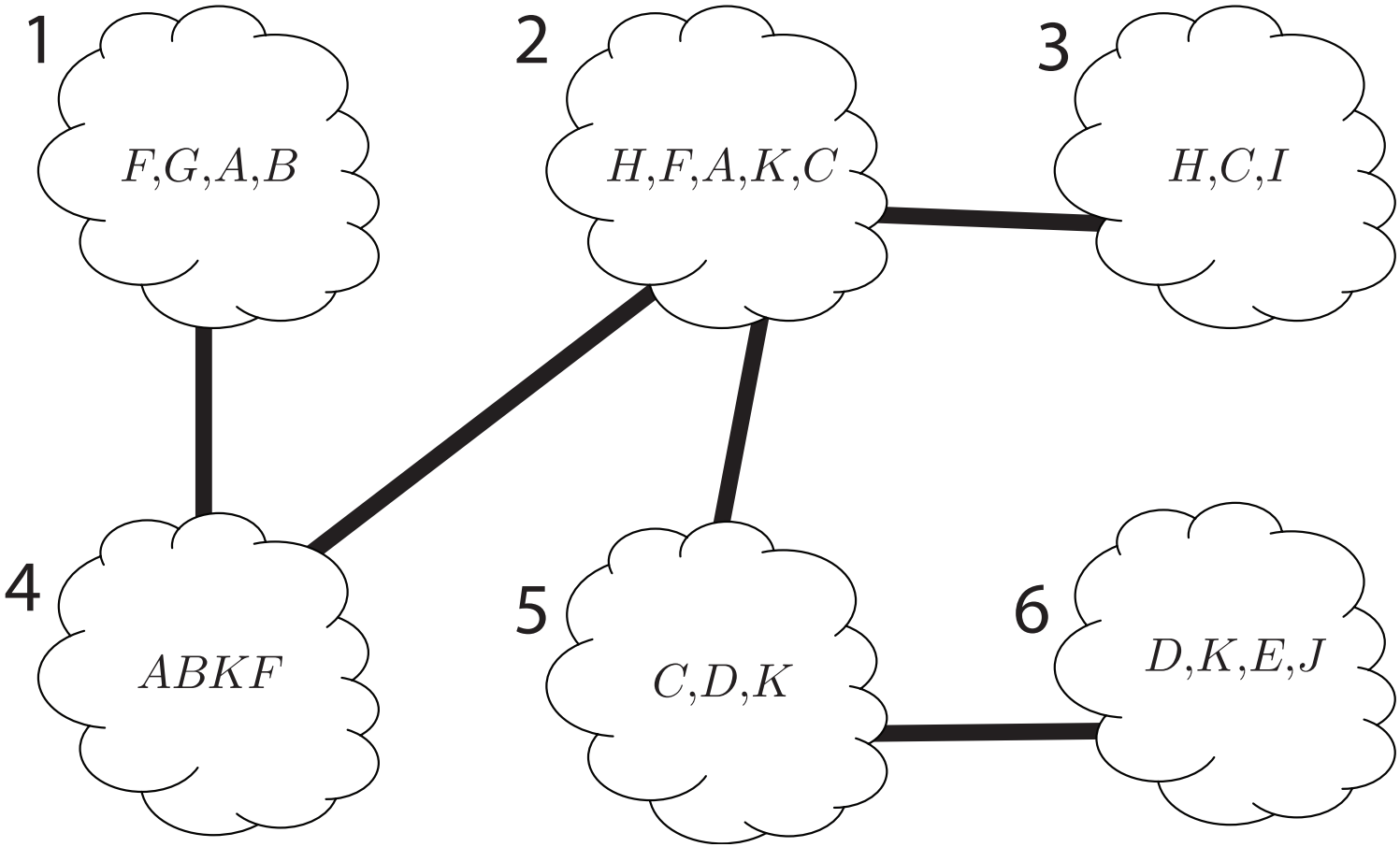
Examples

Cluster Tree that violates the cluster intersection property



Examples

Cluster Tree that obeys the cluster intersection property



Running Intersection Property (r.i.p.)

Definition 6.4.4 (Running Intersection Property (r.i.p.))

Let $C_1, C_2, \ldots, C_\ell$ be an ordered sequence of subsets of $V(G)$. Then the ordering obeys the running intersection property (r.i.p.) property if for all $i > 1$, there exists $j < i$ such that $C_i \cap (\cup_{k<i} C_k) = C_i \cap C_j$.

- Cluster j acts as a representative for all of i's history.

Running Intersection Property (r.i.p.)

Definition 6.4.4 (Running Intersection Property (r.i.p.))

Let $C_1, C_2, \ldots, C_\ell$ be an ordered sequence of subsets of $V(G)$. Then the ordering obeys the running intersection property (r.i.p.) property if for all $i > 1$, there exists $j < i$ such that $C_i \cap (\cup_{k<i} C_k) = C_i \cap C_j$.

- Cluster j acts as a representative for all of i's history.
- r.i.p. is defined in terms of clusters of nodes in a graph.

Running Intersection Property (r.i.p.)

Definition 6.4.4 (Running Intersection Property (r.i.p.))

Let $C_1, C_2, \ldots, C_\ell$ be an ordered sequence of subsets of $V(G)$. Then the ordering obeys the running intersection property (r.i.p.) property if for all $i > 1$, there exists $j < i$ such that $C_i \cap (\cup_{k<i} C_k) = C_i \cap C_j$.

- Cluster j acts as a representative for all of i's history.
- r.i.p. is defined in terms of clusters of nodes in a graph.
- r.i.p. holds on an (unordered) set of clusters if such an ordering can be found.

Running Intersection Property (r.i.p.)

Given sequence of clusters $C_1, C_2, \ldots, C_\ell$. Define the history (accumulation) of sequence at position i:

Running Intersection Property (r.i.p.)

Given sequence of clusters $C_1, C_2, \ldots, C_\ell$. Define the history (accumulation) of sequence at position i:

$$H_i = C_1 \cup C_2 \cup \cdots \cup C_i. \tag{6.4}$$

Running Intersection Property (r.i.p.)

Given sequence of clusters $C_1, C_2, \ldots, C_\ell$. Define the history (accumulation) of sequence at position i:

$$H_i = C_1 \cup C_2 \cup \cdots \cup C_i. \tag{6.4}$$

Innovation (residual) or new nodes in C_i not encountered in the previous history, as:

$$R_i = C_i \setminus H_{i-1}. \tag{6.5}$$

Running Intersection Property (r.i.p.)

Given sequence of clusters $C_1, C_2, \ldots, C_\ell$. Define the history (accumulation) of sequence at position i:

$$H_i = C_1 \cup C_2 \cup \cdots \cup C_i. \tag{6.4}$$

Innovation (residual) or new nodes in C_i not encountered in the previous history, as:

$$R_i = C_i \setminus H_{i-1}. \tag{6.5}$$

Lastly, define the non-innovation, commonality, or separation elements between new and previous history:

$$S_i = C_i \cap H_{i-1} \tag{6.6}$$

Running Intersection Property (r.i.p.)

Given sequence of clusters $C_1, C_2, \ldots, C_\ell$. Define the history (accumulation) of sequence at position i:

$$H_i = C_1 \cup C_2 \cup \cdots \cup C_i. \tag{6.4}$$

Innovation (residual) or new nodes in C_i not encountered in the previous history, as:

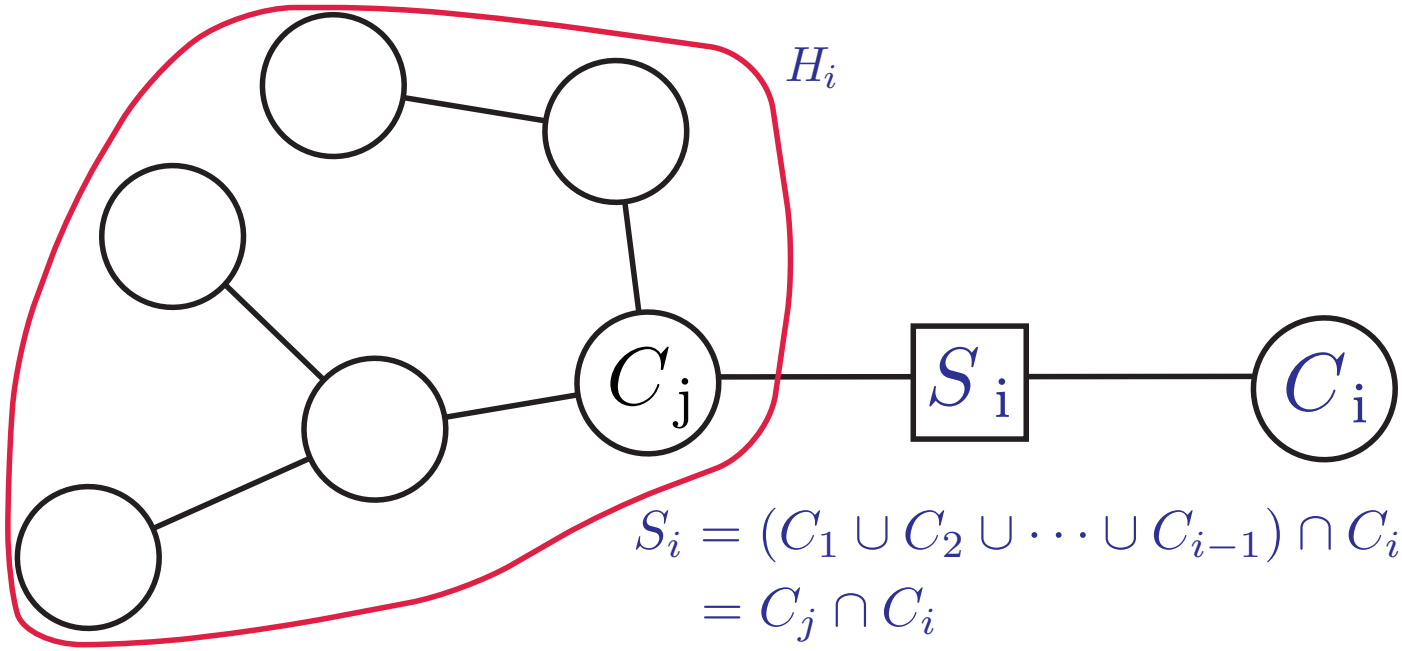
$$R_i = C_i \setminus H_{i-1}. \tag{6.5}$$

Lastly, define the non-innovation, commonality, or separation elements between new and previous history:

$$S_i = C_i \cap H_{i-1} \tag{6.6}$$

Note $C_i = R_i \cup S_i$, i^{th} clusters consists of the innovation R_i and the commonality S_i.

Running Intersection Property (r.i.p.)



Clusters are in r.i.p. order if the commonality S_i between new and history is fully contained in one element of history. I.e., there exists an $j < i$ such that $S_i \subseteq C_j$.

First Two Properties

Lemma 6.4.5

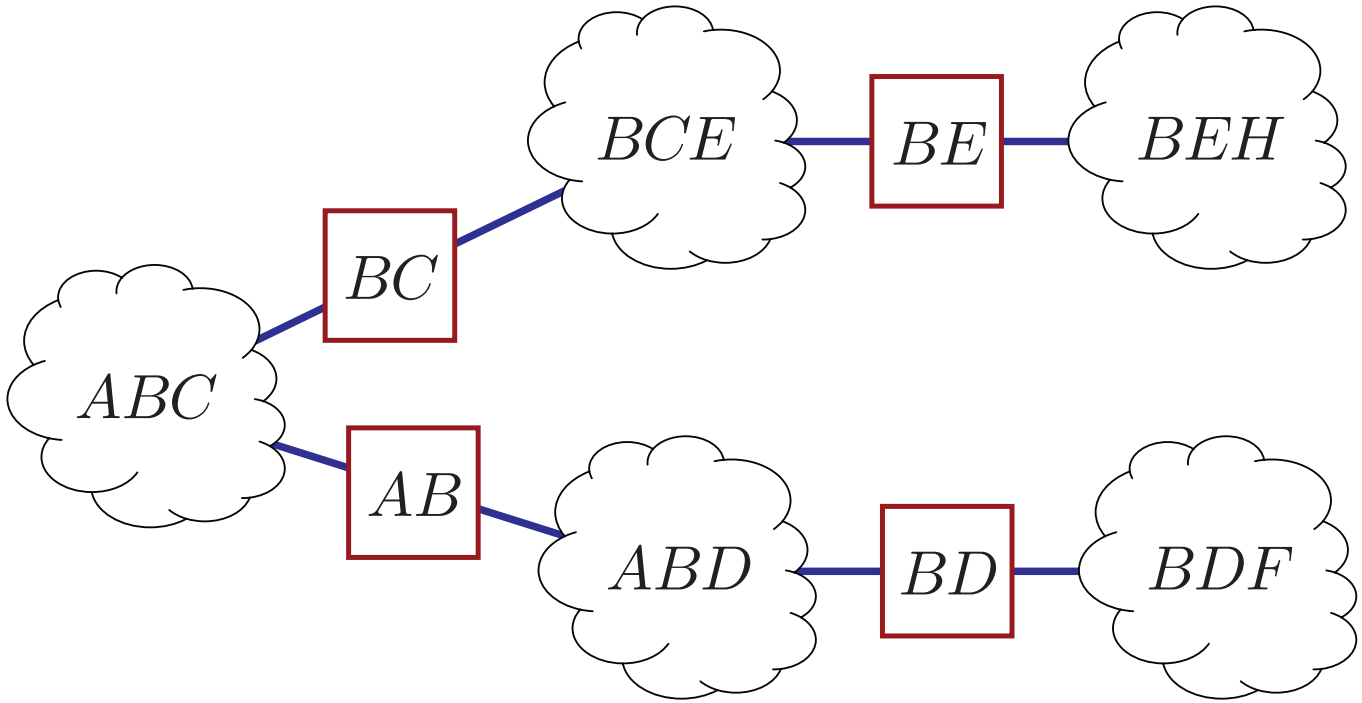
The cluster intersection and running intersection properties are identical.

Proof.

Starting with clusters in r.i.p. order, construct cluster tree by connecting each i to its corresponding j node. This is a tree. Also, take any C_i, C_k with $k < i$. S_i summarizes everything between C_i and H_{i-1} so $C_i \cap C_k \subseteq S_i$. Apply recursively on unique path between C_i and C_k. Conversely, perform traversal (depth or breadth first search) on cluster tree. That order will satisfy r.i.p. since any possible intersection between C_i, C_j on unique path, it must be fully contained in neighbor.

□

First Two Properties



Example of a set of node clusters (within the cloud-like shapes) arranged in a tree that satisfies the r.i.p. and also the cluster intersection property. The intersections between neighboring node clusters are shown in the figure as square boxes. Consider the path or $\{B, E, H\} \cap \{B, D, F\} = \{B\}$.

Induced sub-tree property (i.s.p.)

Definition 6.4.6 (Induced Sub-tree Property)

Given a cluster tree $\mathcal{T}$ for graph G, the *induced sub-tree property* holds for $\mathcal{T}$ if for all $v \in V$, the set of clusters $C \in \mathcal{C}$ such that $v \in C$ induces a sub-tree $\mathcal{T}(v)$ of $\mathcal{T}$.

Note, by definition the sub-tree is necessarily connected.

Three properties

Lemma 6.4.7

Induced sub-tree property holds iff cluster intersection property holds

Proof.

Assume induced subtree holds. Take all $v \in C_i \cap C_j$, then each such v induces a sub-tree of $\mathcal{T}$, and all of these sub-trees overlap on the unique path between C_i and C_j in $\mathcal{T}$.
Conversely, when cluster intersection property holds, given $v \in V$, consider all clusters that contain v, $\mathcal{C}(v) = \{C \in \mathcal{C} : v \in C\}$. For any pair $C_1, C_2 \in \mathcal{C}(v)$, we have that $C_1 \cap C_2$ exists on the unique path between C_1 and C_2 in $\mathcal{T}$, and since $v \in C_1 \cap C_2$, v always exists on each of these paths. These paths, considered as a union together, cannot form a cycle (since they are paths on a tree). Moreover, these paths unioned together form a tree (they're connected).

□

Therefore, cluster intersection property, running intersection property, and induced sub-tree property, are all identical. We'll henceforth refer them collectively as r.i.p.

Tree decomposition

Lets look again at tree decomposition, a cluster tree that satisfies (what we now know to be the) induced sub-tree property (e.g., r.i.p. and c.i.p. as well).

Definition 6.4.8 (tree decomposition)

Given a graph $G = (V, E)$, a tree-decomposition of a graph is a pair $(\{C_i : i \in I\}, T)$ where $\mathcal{T} = (I, \mathcal{E}_T)$ is a tree with node index set I, edge set $\mathcal{E}_T$, and $\{C_i\}_i$ (one for each $i \in I$) is a collection of clusters (subsets) of $V(G)$ such that:

1. $\cup_{i \in I} C_i = V$
2. for any edge $(u, v) \in E(G)$, there exists $i \in I$ with $u, v \in C_i$
3. (r.i.p.) for any $v \in V$, the set $\{i \in I : v \in C_i\}$ forms a (nec. connected) subtree of T

Recap

- We want all original graph (o.g.) clique marginals. Why?

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.
- Def: decomposition of a graph, and factorization implication.

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.
- Def: decomposition of a graph, and factorization implication.
- Def: decomposable graph, and decomposition tree

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.
- Def: decomposition of a graph, and factorization implication.
- Def: decomposable graph, and decomposition tree
- Thm: triangulated graph $\equiv$ decomposable graph

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.
- Def: decomposition of a graph, and factorization implication.
- Def: decomposable graph, and decomposition tree
- Thm: triangulated graph $\equiv$ decomposable graph
- Def: tree decomposition (vertex and edge cover, and induced sub-tree).

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.
- Def: decomposition of a graph, and factorization implication.
- Def: decomposable graph, and decomposition tree
- Thm: triangulated graph $\equiv$ decomposable graph
- Def: tree decomposition (vertex and edge cover, and induced sub-tree).
- Def: cluster graph, cluster tree, based only on o.g. nodes, not o.g. edges. Edges in cluster graph cluster tree via cluster intersection.

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.
- Def: decomposition of a graph, and factorization implication.
- Def: decomposable graph, and decomposition tree
- Thm: triangulated graph $\equiv$ decomposable graph
- Def: tree decomposition (vertex and edge cover, and induced sub-tree).
- Def: cluster graph, cluster tree, based only on o.g. nodes, not o.g. edges. Edges in cluster graph cluster tree via cluster intersection.
- Def: cluster intersection property, running intersection property, induced sub-tree property, r.i.p.

Recap

- We want all original graph (o.g.) clique marginals. Why?
- Finding optimal elimination order is optimal for all o.g. clique marginals.
- Def: decomposition of a graph, and factorization implication.
- Def: decomposable graph, and decomposition tree
- Thm: triangulated graph $\equiv$ decomposable graph
- Def: tree decomposition (vertex and edge cover, and induced sub-tree).
- Def: cluster graph, cluster tree, based only on o.g. nodes, not o.g. edges. Edges in cluster graph cluster tree via cluster intersection.
- Def: cluster intersection property, running intersection property, induced sub-tree property, r.i.p.
- Next def: Junction tree, cluster tree with r.i.p. and edge cover.

Junction Tree

Definition 6.4.9

Given a graph $G = (V, E)$, a junction tree corresponding to G (if it exists) is a cluster tree $\mathcal{T} = (\mathcal{C}, E_T)$ having the r.i.p. over the clusters, and where any nodes u, v adjacent via edge $(u, v) \in E(G)$ are together in at least one cluster.

Junction Tree

Definition 6.4.9

Given a graph $G = (V, E)$, a junction tree corresponding to G (if it exists) is a cluster tree $\mathcal{T} = (\mathcal{C}, E_T)$ having the r.i.p. over the clusters, and where any nodes u, v adjacent via edge $(u, v) \in E(G)$ are together in at least one cluster.

- So, junction tree (JT), for a given graph G, is a cluster tree that: 1) satisfies r.i.p. over the clusters, and 2) includes all edges (edge cover). Not all r.i.p.-satisfying cluster trees need be an edge cover.

Junction Tree

Definition 6.4.9

Given a graph $G = (V, E)$, a junction tree corresponding to G (if it exists) is a cluster tree $\mathcal{T} = (\mathcal{C}, E_T)$ having the r.i.p. over the clusters, and where any nodes u, v adjacent via edge $(u, v) \in E(G)$ are together in at least one cluster.

- So, junction tree (JT), for a given graph G, is a cluster tree that: 1) satisfies r.i.p. over the clusters, and 2) includes all edges (edge cover). Not all r.i.p.-satisfying cluster trees need be an edge cover.
- Clusters in JT need not be original graph cliques!!

Junction Tree

Definition 6.4.9

Given a graph $G = (V, E)$, a junction tree corresponding to G (if it exists) is a cluster tree $\mathcal{T} = (\mathcal{C}, E_T)$ having the r.i.p. over the clusters, and where any nodes u, v adjacent via edge $(u, v) \in E(G)$ are together in at least one cluster.

- So, junction tree (JT), for a given graph G, is a cluster tree that: 1) satisfies r.i.p. over the clusters, and 2) includes all edges (edge cover). Not all r.i.p.-satisfying cluster trees need be an edge cover.
- Clusters in JT need not be original graph cliques!!
- JT could have clusters corresponding to cliques, maxcliques, or neither of the above.

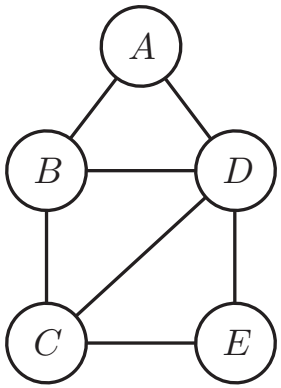
Junction Tree

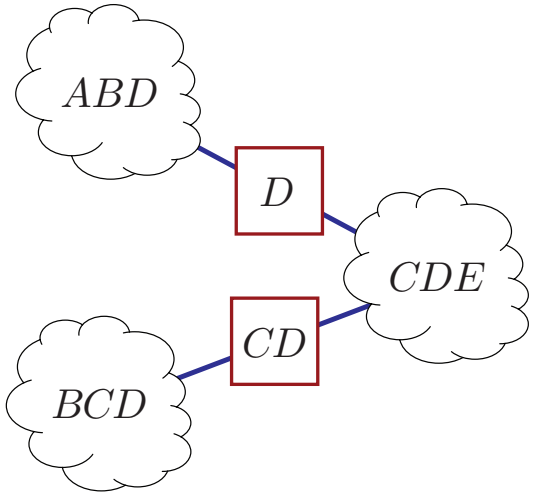
Definition 6.4.9

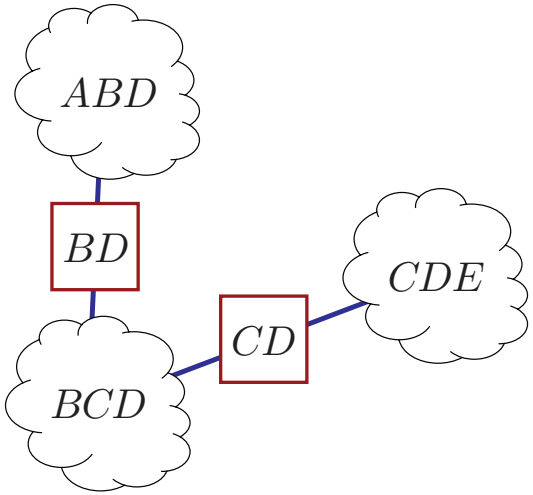
Given a graph $G = (V, E)$, a junction tree corresponding to G (if it exists) is a cluster tree $\mathcal{T} = (\mathcal{C}, E_T)$ having the r.i.p. over the clusters, and where any nodes u, v adjacent via edge $(u, v) \in E(G)$ are together in at least one cluster.

- So, junction tree (JT), for a given graph G, is a cluster tree that: 1) satisfies r.i.p. over the clusters, and 2) includes all edges (edge cover). Not all r.i.p.-satisfying cluster trees need be an edge cover.
- Clusters in JT need not be original graph cliques!!
- JT could have clusters corresponding to cliques, maxcliques, or neither of the above.
- If clusters correspond to the original graph cliques (resp. maxcliques) in G, it called a junction tree of cliques (resp. maxcliques).

Examples junction trees and not

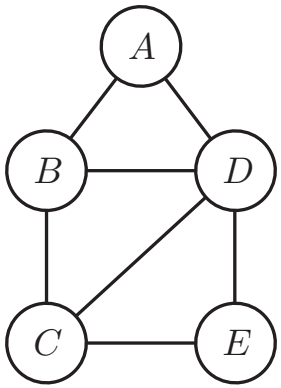


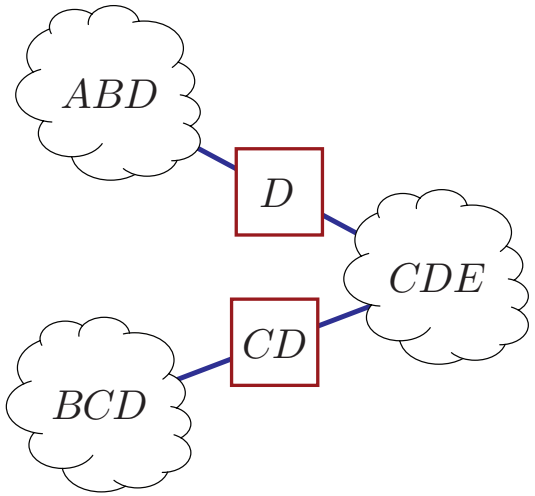


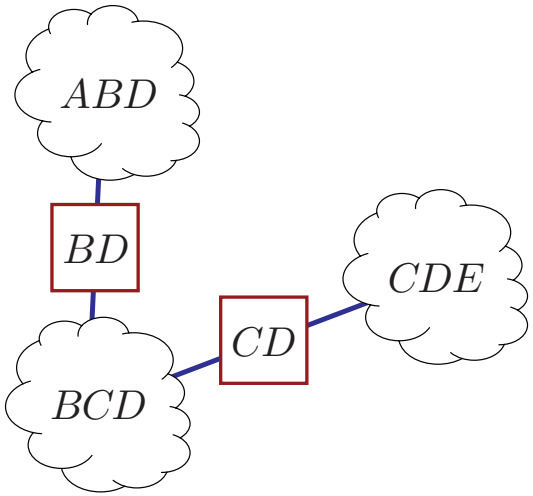


Questions to ask:

Examples junction trees and not



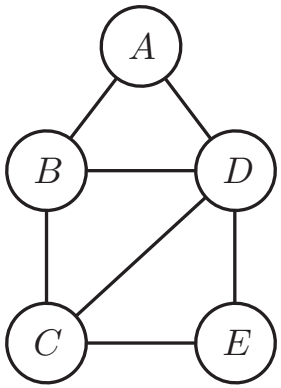


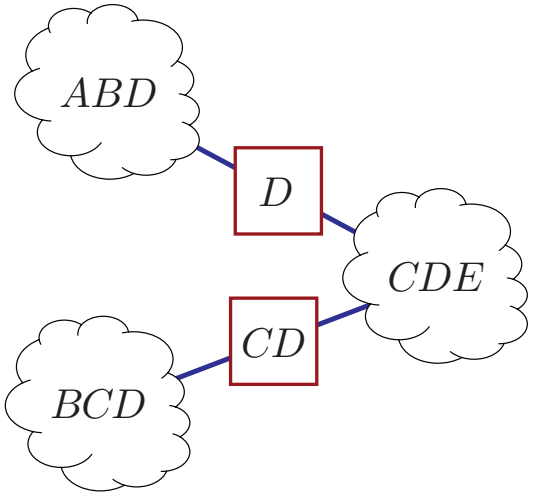


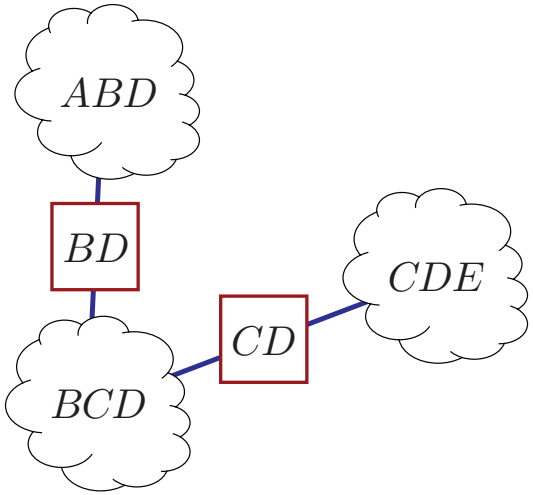
Questions to ask:

- cluster graph?

Examples junction trees and not



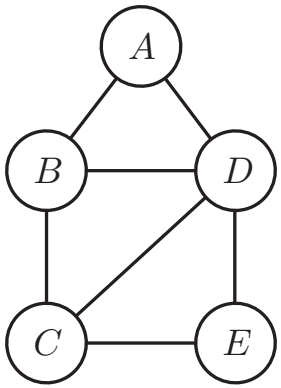


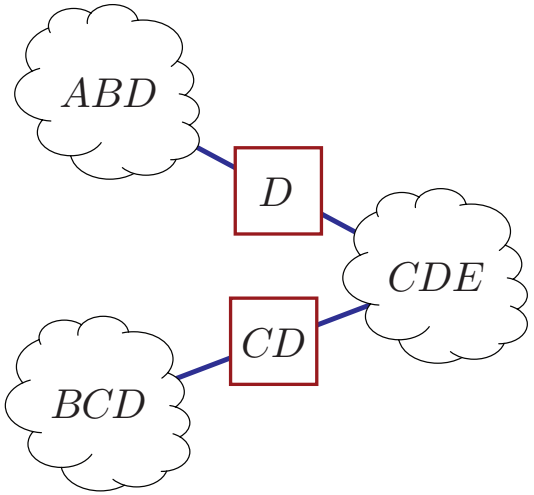


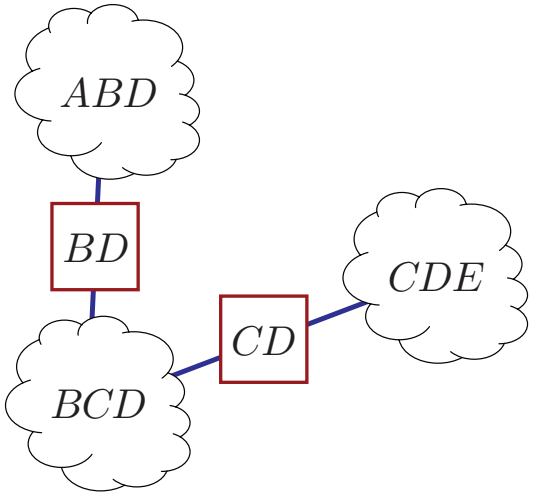
Questions to ask:

- cluster graph?
- cluster tree?

Examples junction trees and not



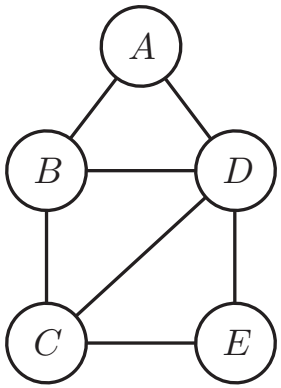


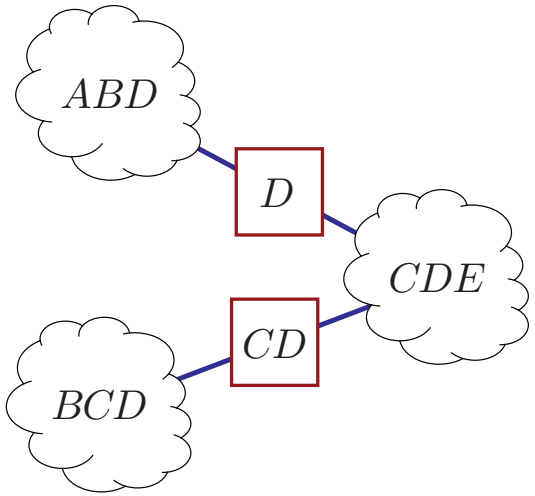


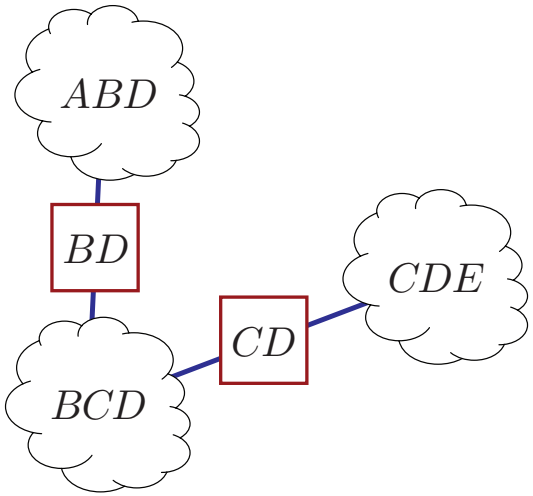
Questions to ask:

- cluster graph?
- cluster tree?
- Junction tree?

Examples junction trees and not



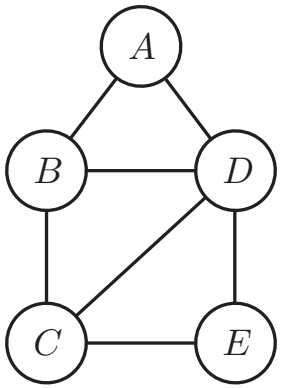


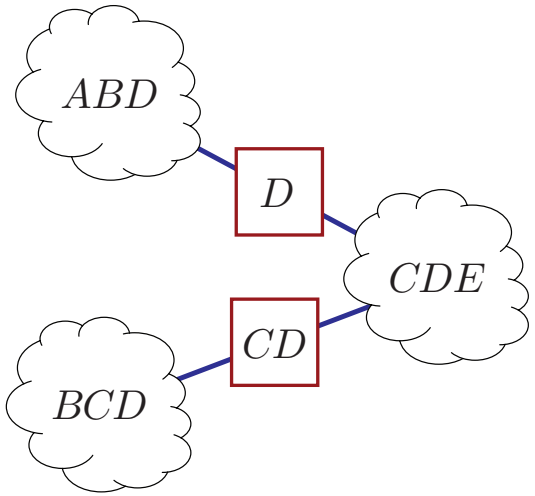


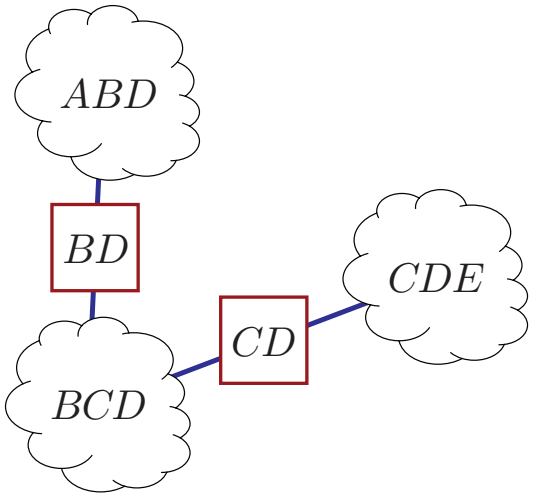
Questions to ask:

- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?

Examples junction trees and not



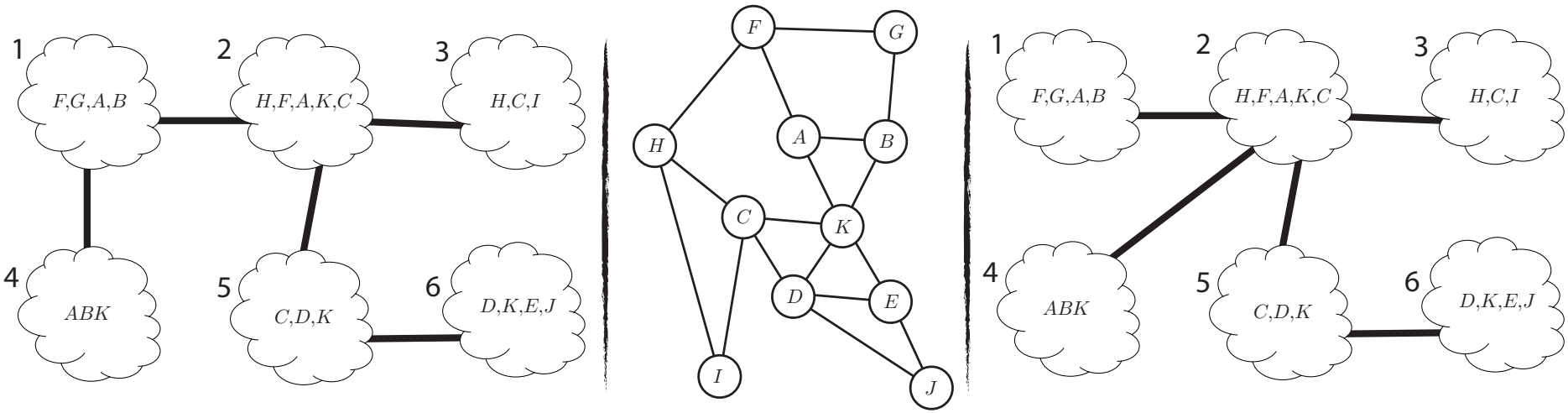




Questions to ask:

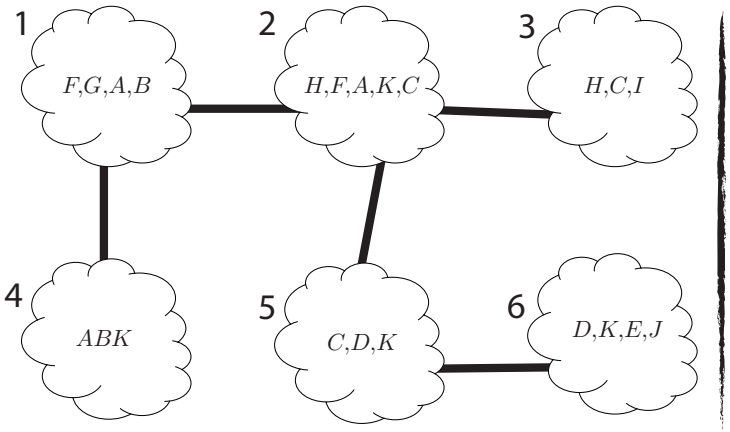
- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?
- Junction tree of maxcliques?

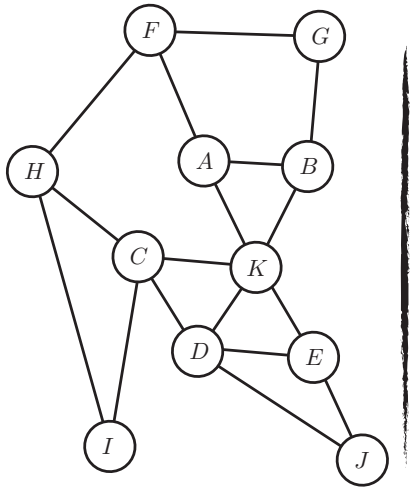
Examples junction trees and not

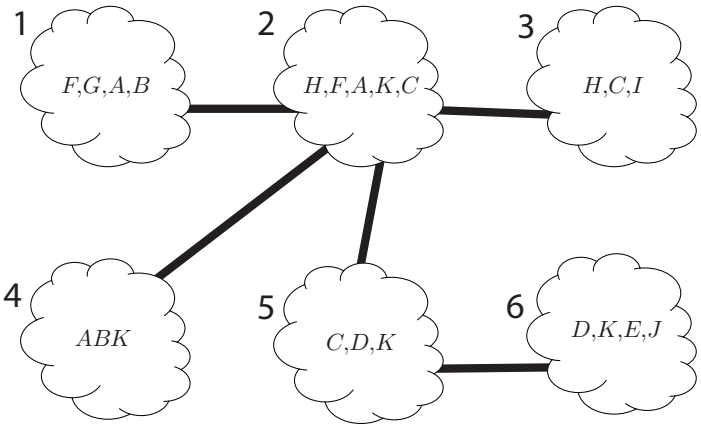


- cluster graph?

Examples junction trees and not

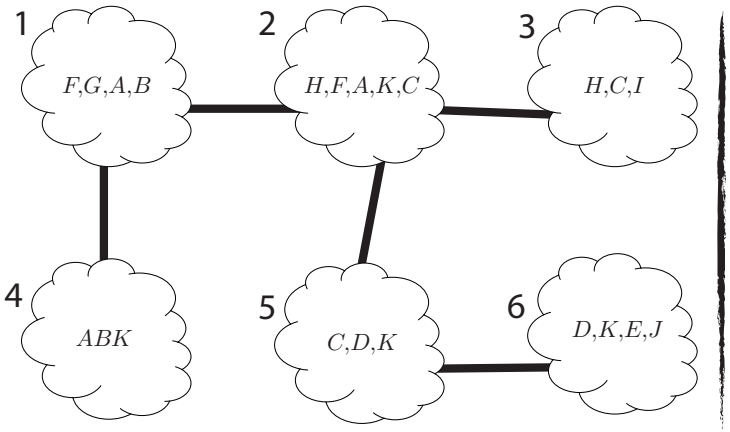


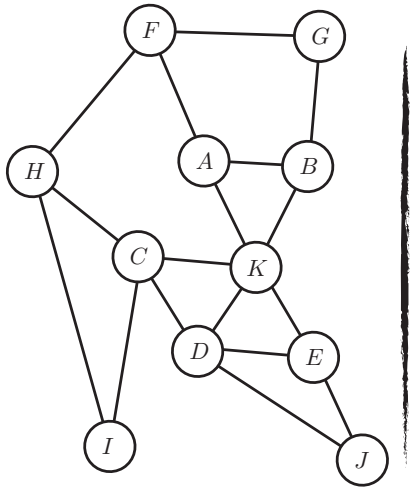


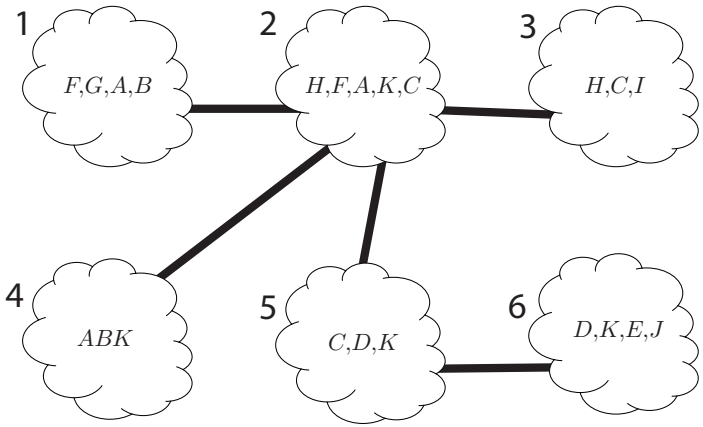


- cluster graph?
- cluster tree?

Examples junction trees and not

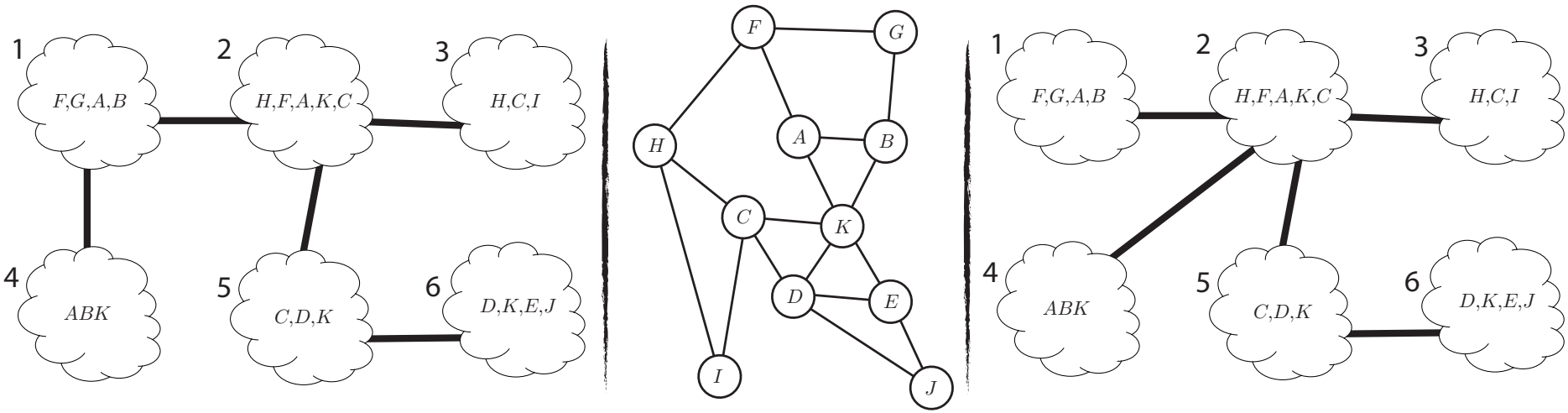






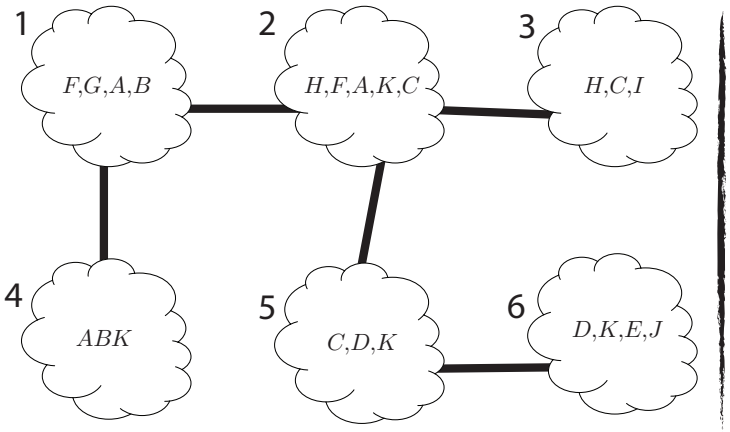
- cluster graph?
- cluster tree?
- Junction tree?

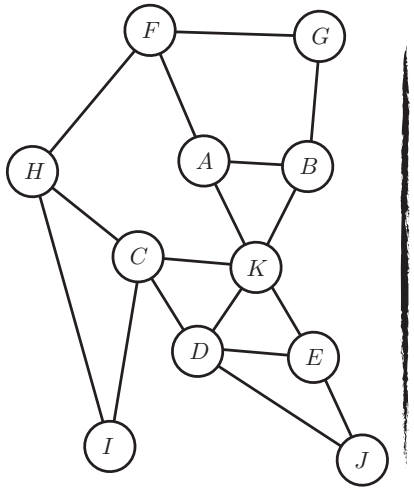
Examples junction trees and not

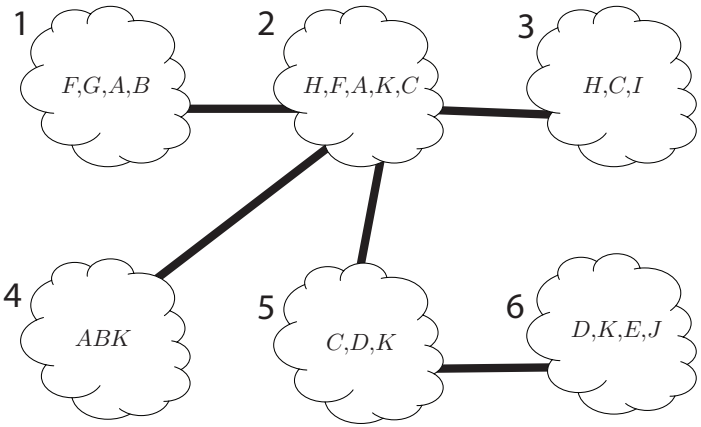


- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?

Examples junction trees and not

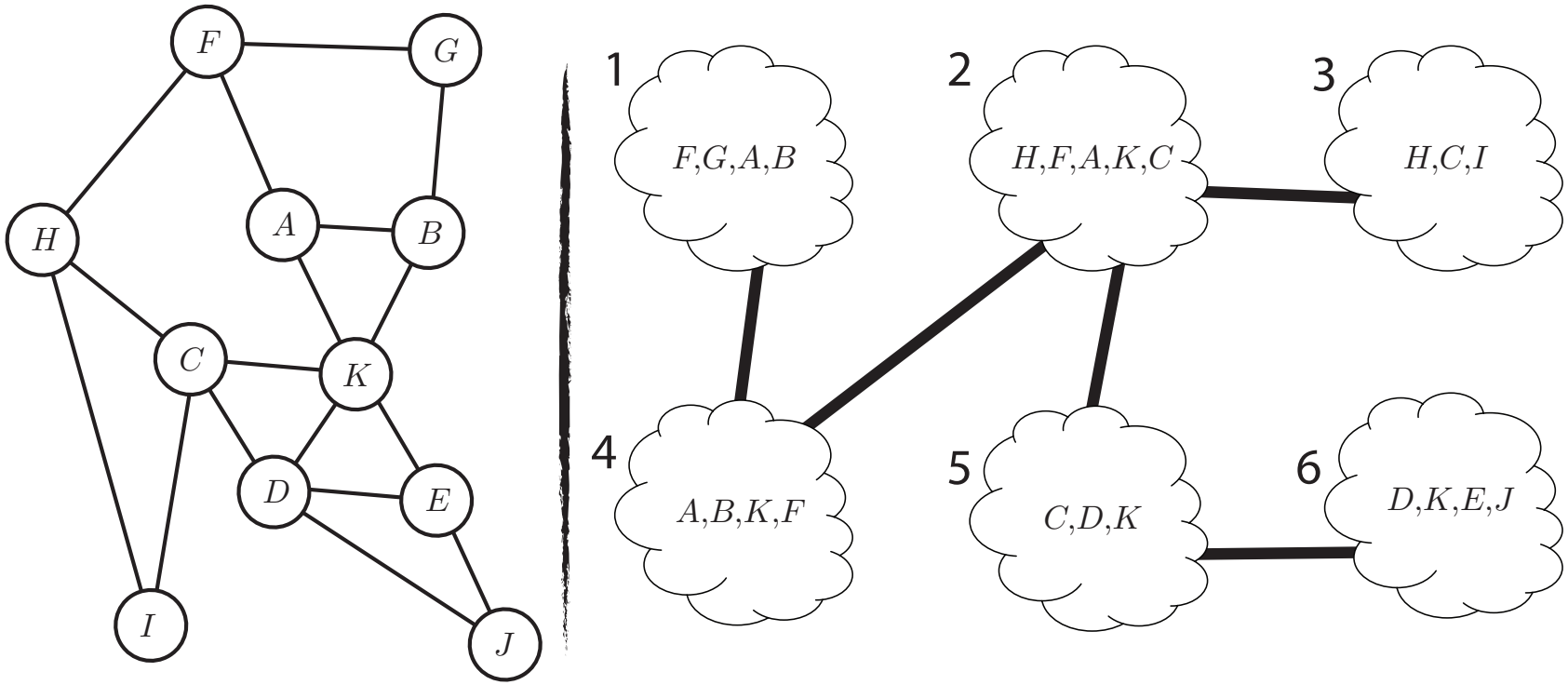






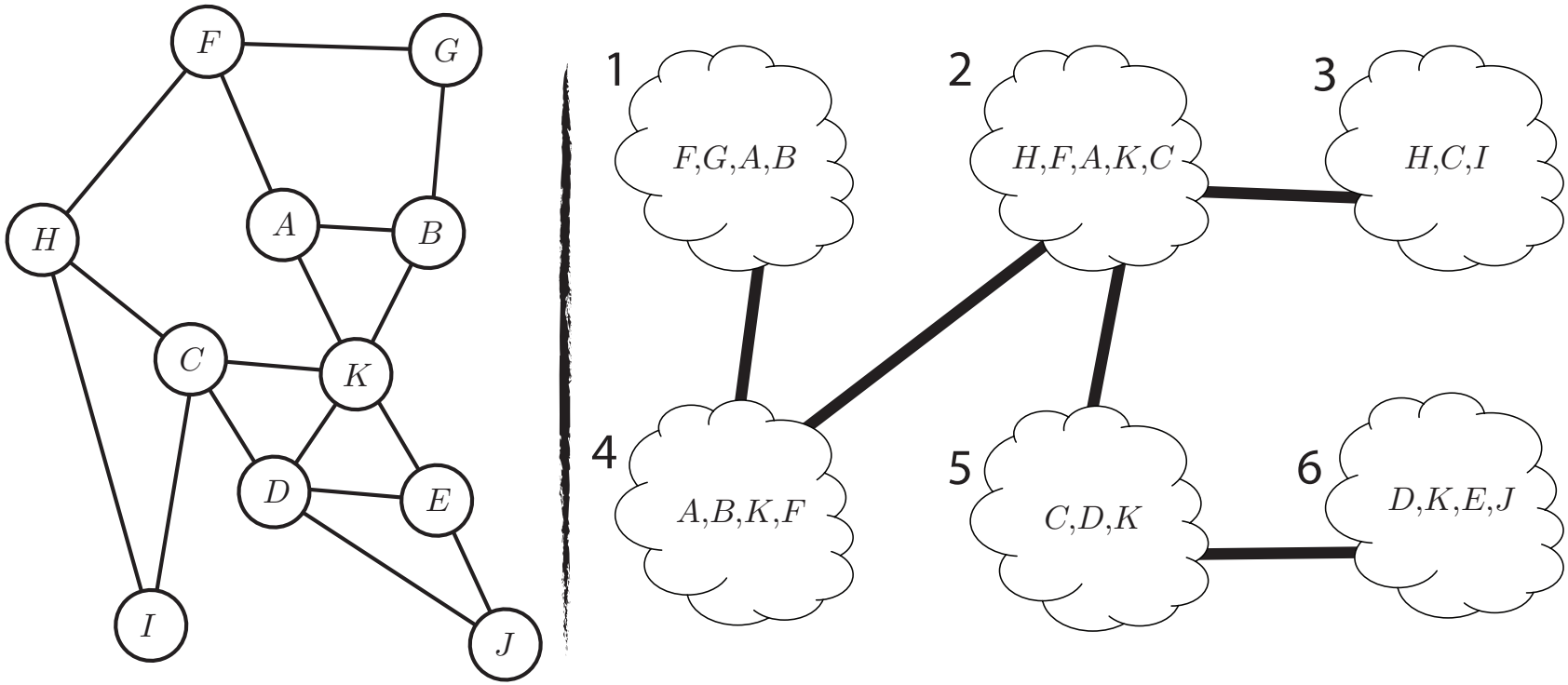
- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?
- Junction tree of maxcliques?

Examples junction trees and not



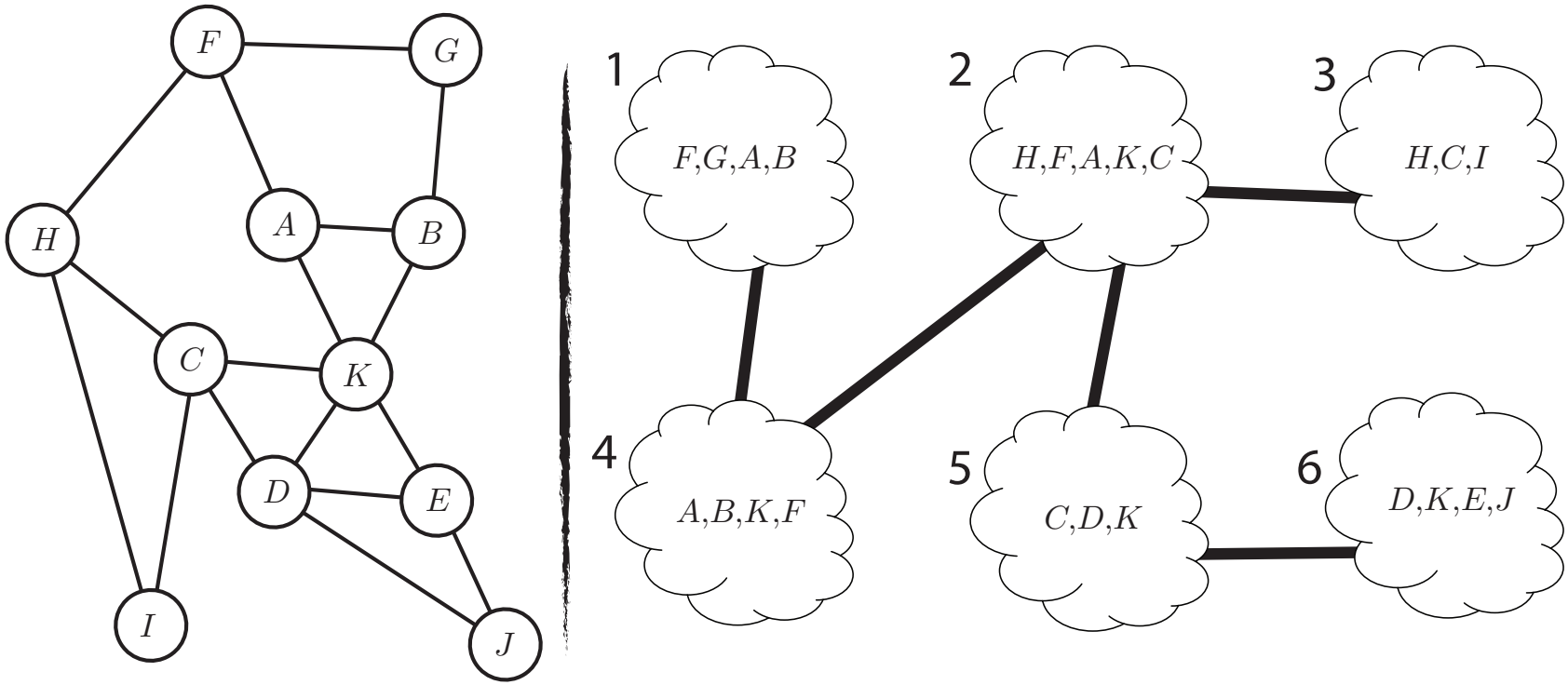
- cluster graph?

Examples junction trees and not



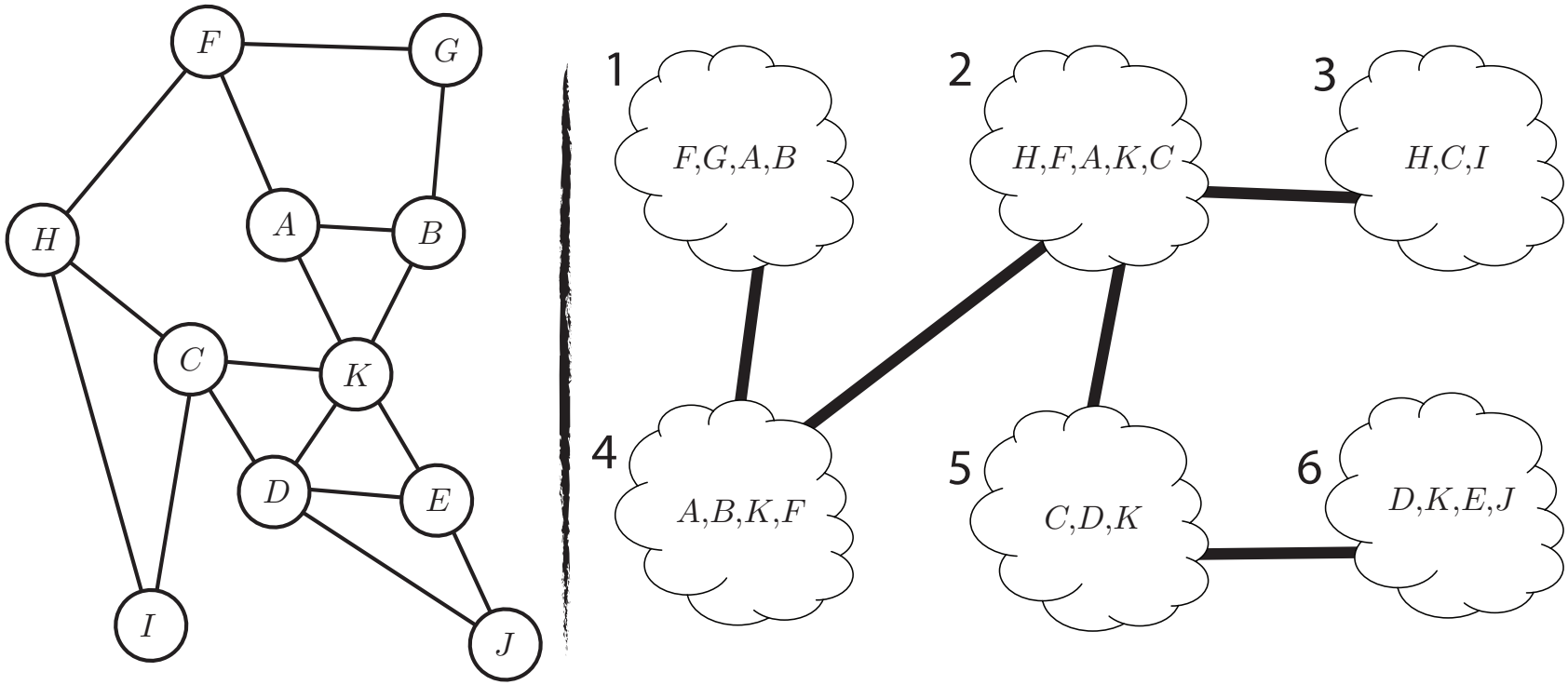
- cluster graph?
- cluster tree?

Examples junction trees and not



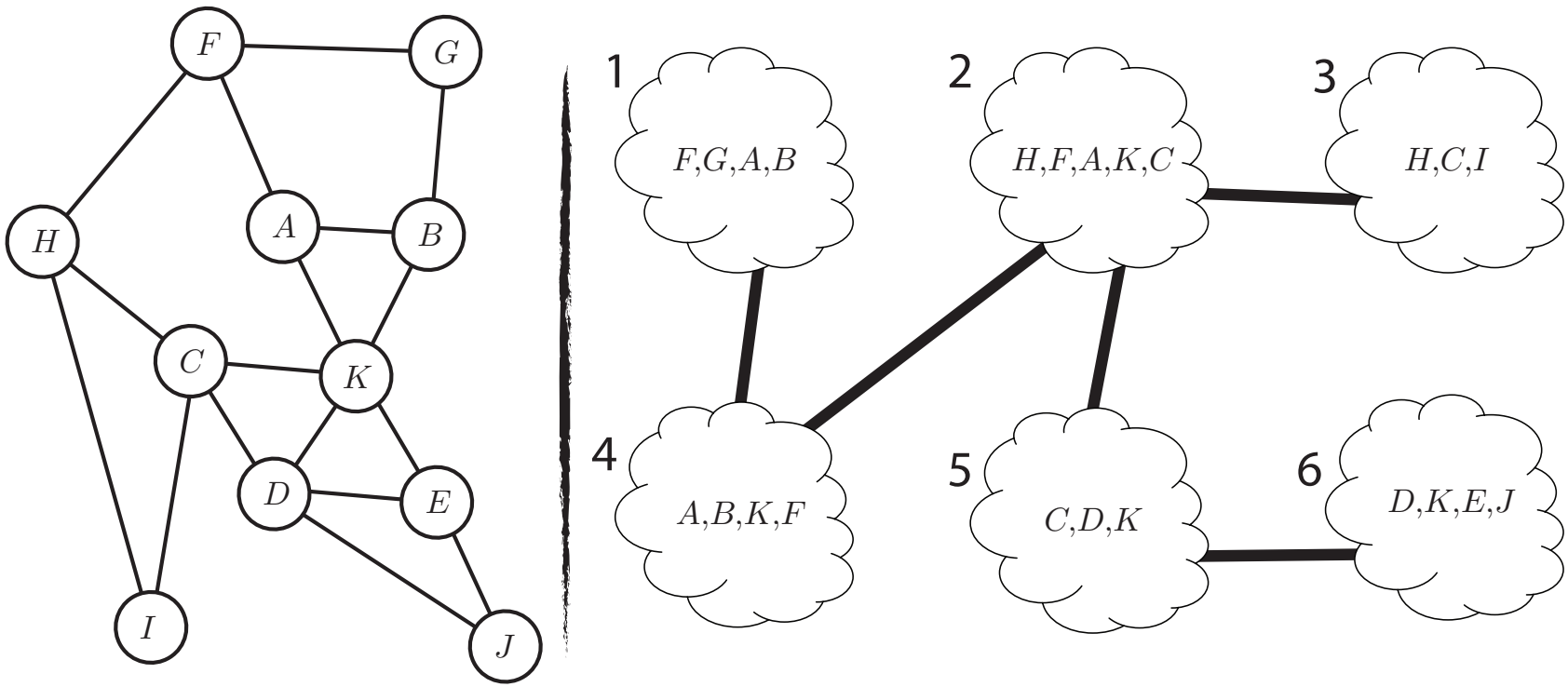
- cluster graph?
- cluster tree?
- Junction tree?

Examples junction trees and not



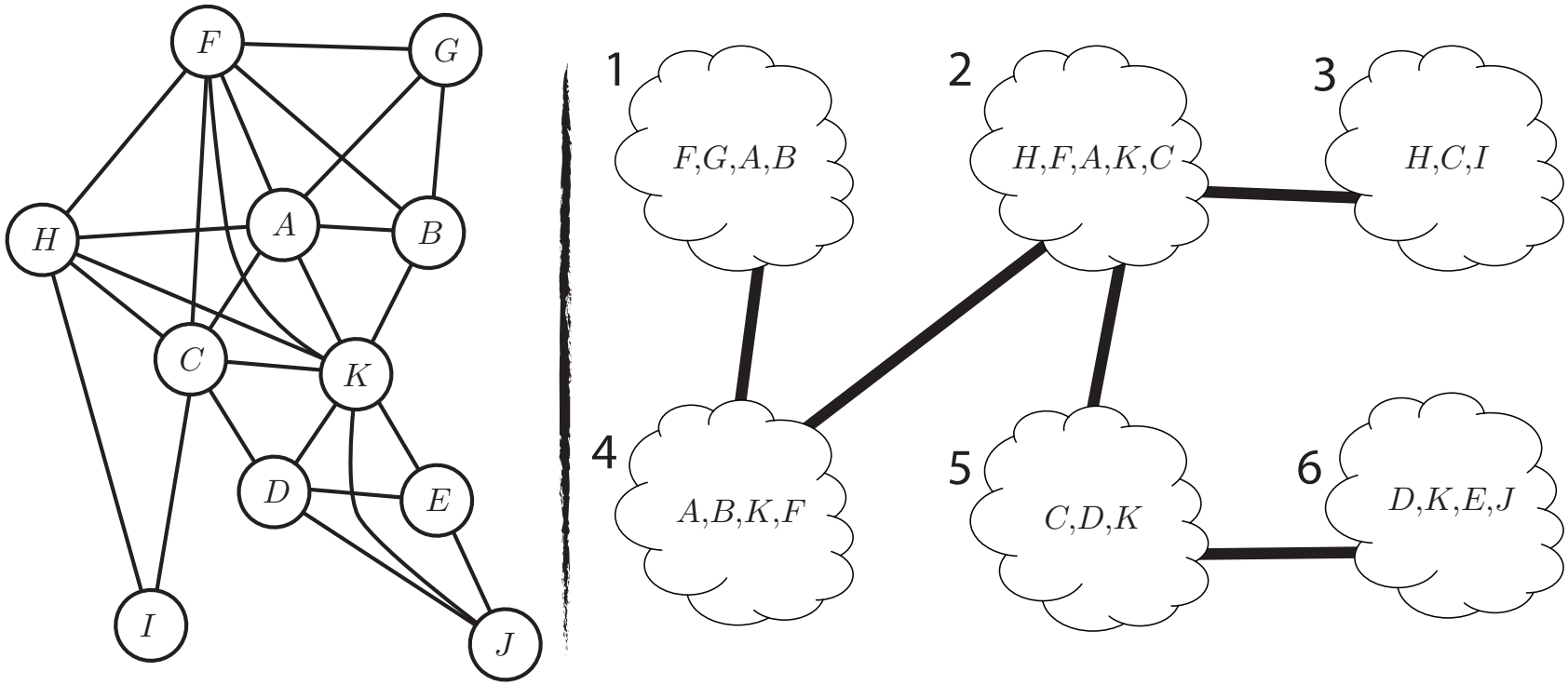
- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?

Examples junction trees and not



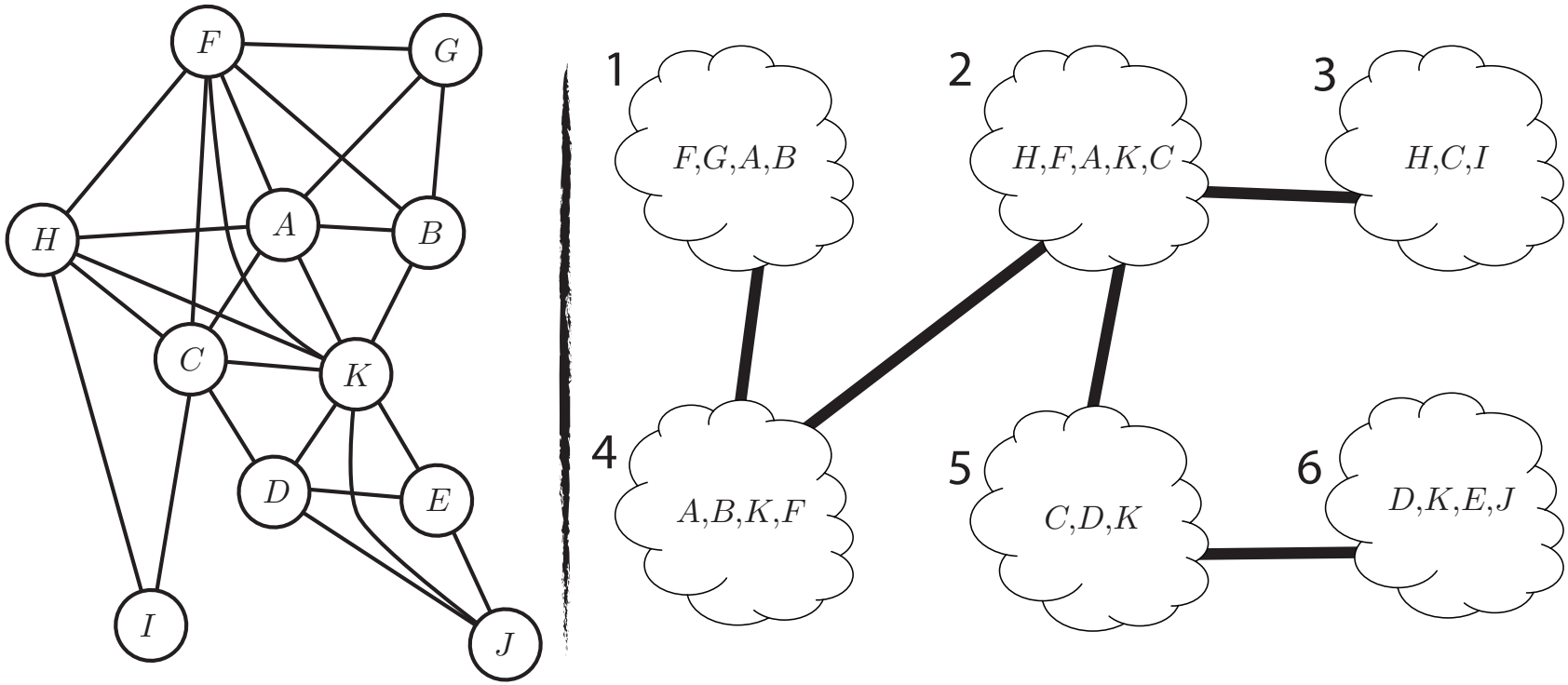
- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?
- Junction tree of maxcliques?

Examples junction trees and not



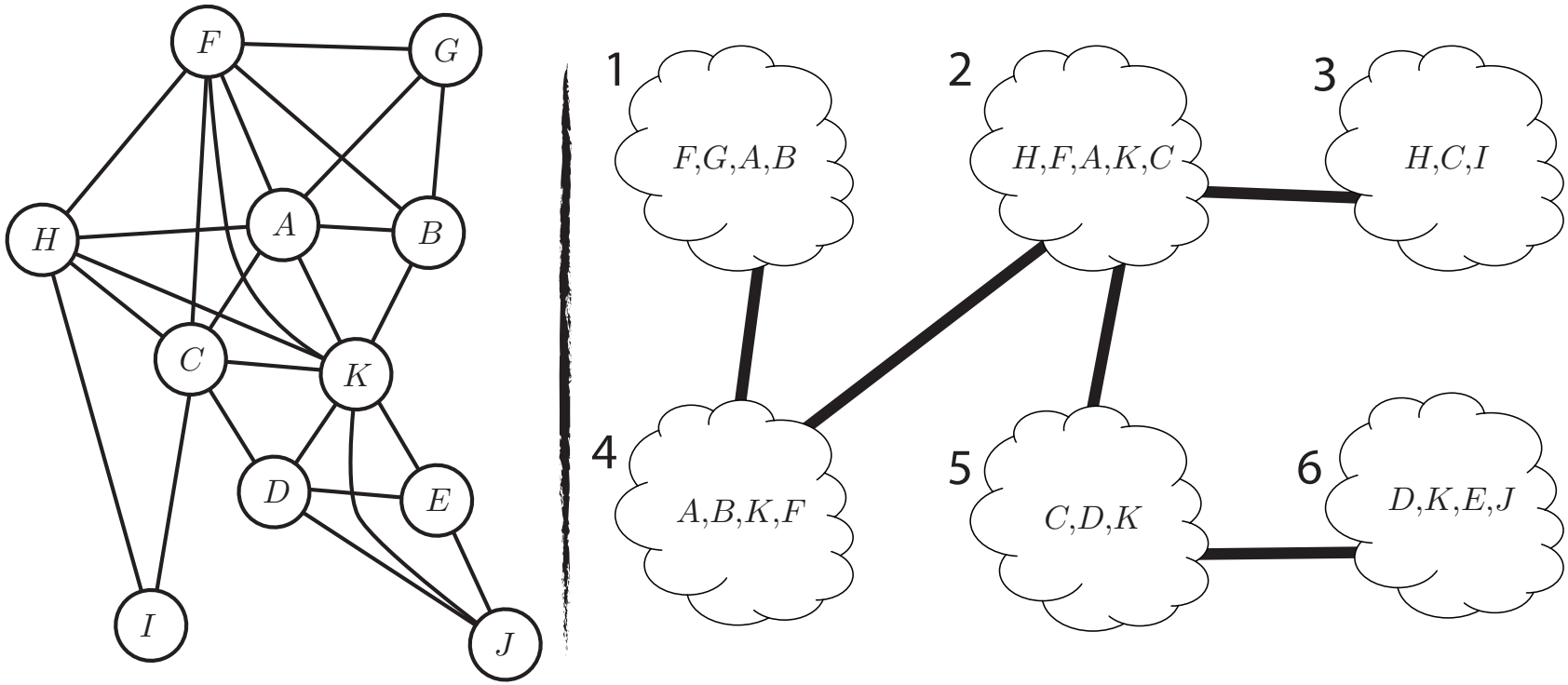
- cluster graph?

Examples junction trees and not



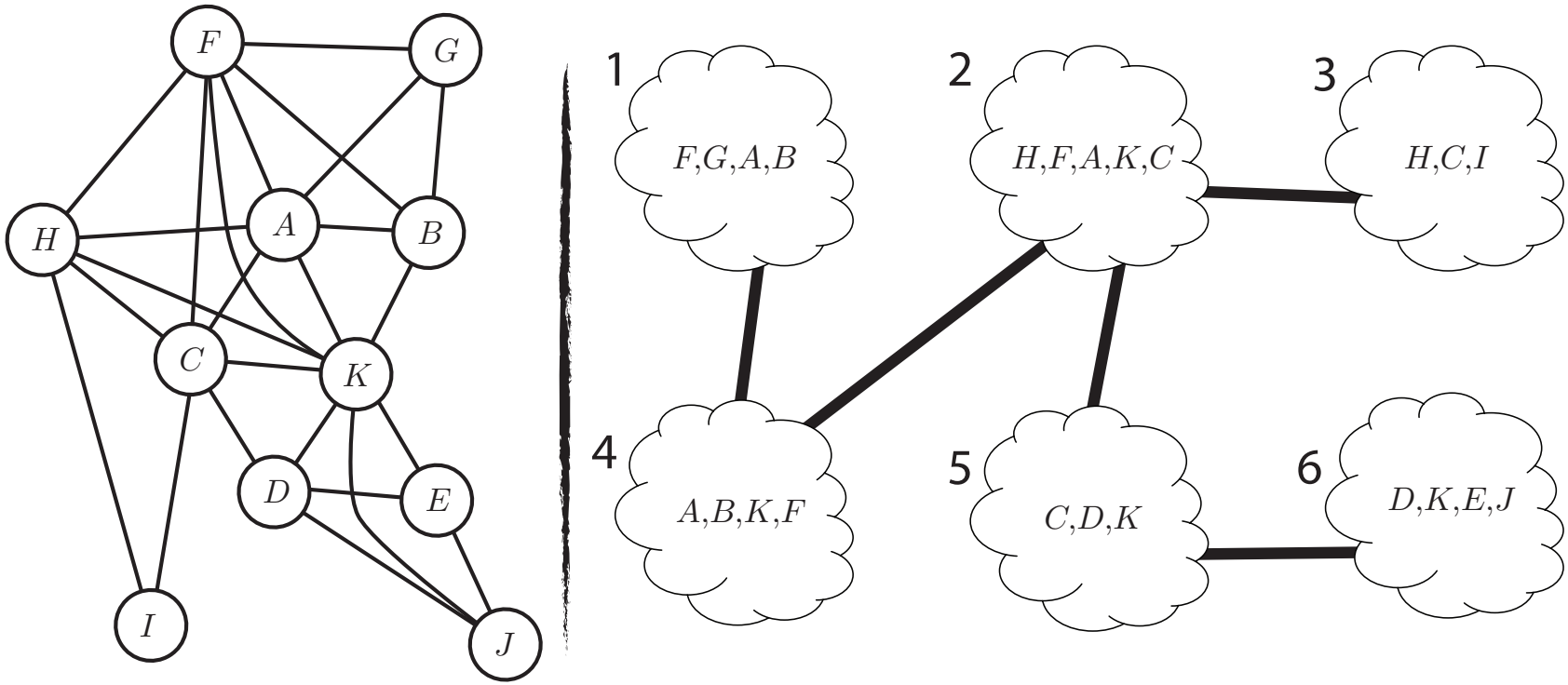
- cluster graph?
- cluster tree?

Examples junction trees and not



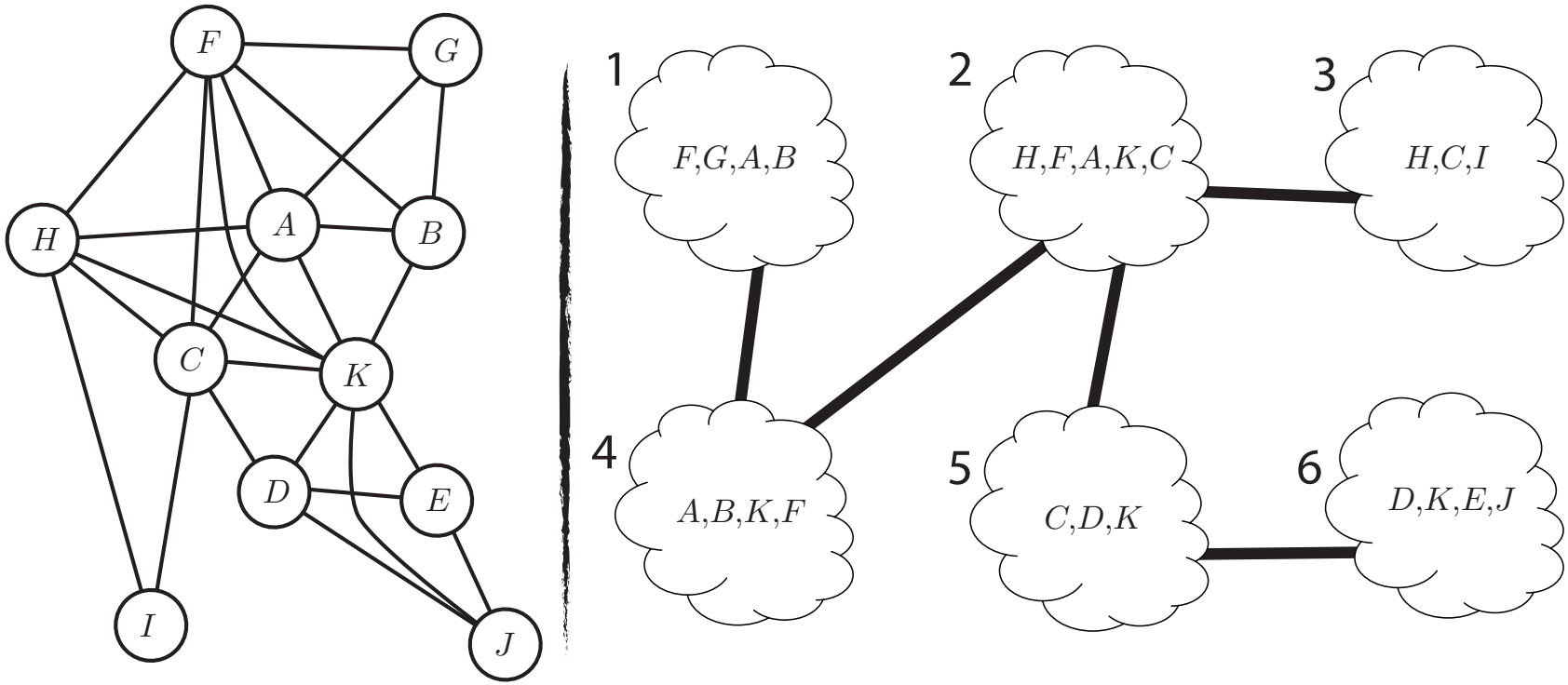
- cluster graph?
- cluster tree?
- Junction tree?

Examples junction trees and not



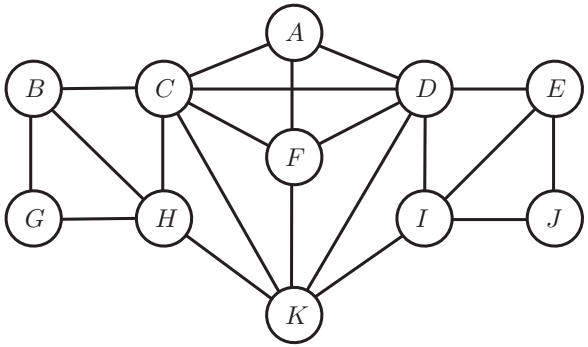
- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?

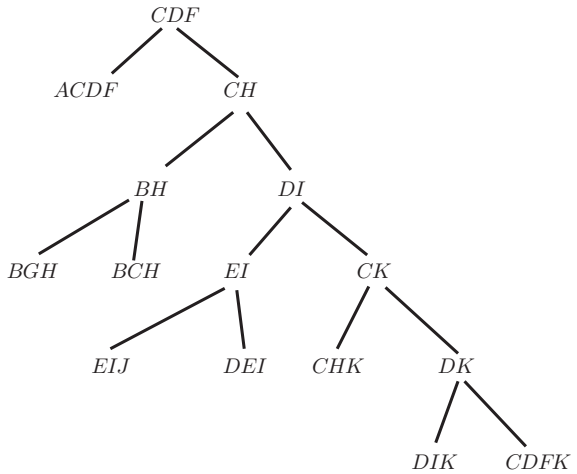
Examples junction trees and not

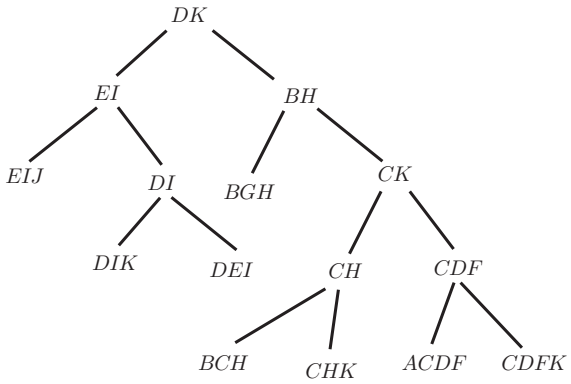


- cluster graph?
- cluster tree?
- Junction tree?
- Junction tree of cliques?
- Junction tree of maxcliques?

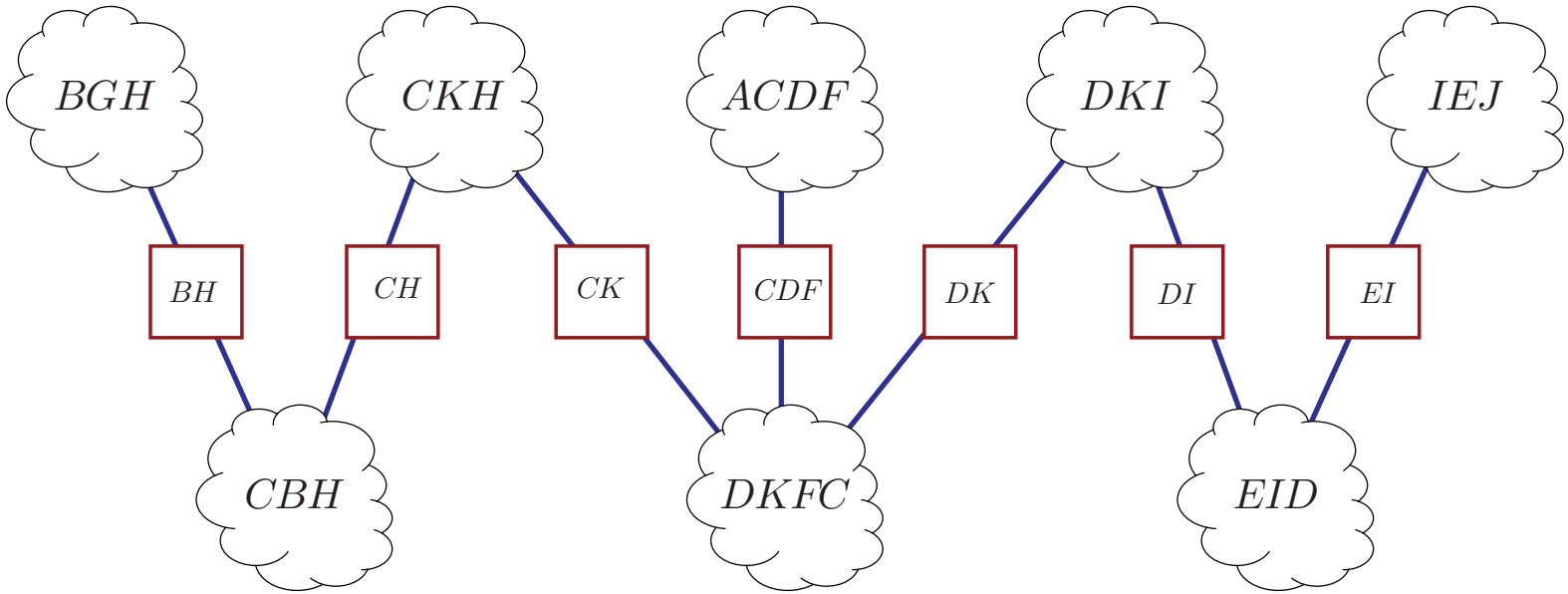
Examples junction trees and not







Tree of cliques for above graph.Does r.i.p. hold?



Junction Tree Preserving Operations

Lemma 6.4.10

Given a junction tree, form a new cluster tree as follows. For each cluster C in the JT, choose an order of nodes within C, say $c_1, c_2, \ldots, c_k$, and hang a chain of clusters off of C consisting of $C \setminus \{c_1\}$ hanging from C, $C \setminus \{c_1, c_2\}$ hanging from $C \setminus \{c_1\}$, $C \setminus \{c_1, c_2, c_3\}$ hanging from $C \setminus \{c_1, c_2\}$, and so on. Then the resulting cluster graph is a cluster tree, and moreover it is still junction tree.

Junction Tree Preserving Operations

Lemma 6.4.10

Given a junction tree, form a new cluster tree as follows. For each cluster C in the JT, choose an order of nodes within C, say $c_1, c_2, \ldots, c_k$, and hang a chain of clusters off of C consisting of $C \setminus \{c_1\}$ hanging from C, $C \setminus \{c_1, c_2\}$ hanging from $C \setminus \{c_1\}$, $C \setminus \{c_1, c_2, c_3\}$ hanging from $C \setminus \{c_1, c_2\}$, and so on. Then the resulting cluster graph is a cluster tree, and moreover it is still junction tree.

Lemma 6.4.11

Given a junction tree, where (C_i, C_j) are neighboring clusters in the tree, we can merge these two clusters forming a new cluster $C_{ij} = C_i \cup C_j$, and where the neighbors of C_{ij} are the set of neighbors of either C_i and C_j. Then the resulting structure is still junction tree.

Junction Tree Preserving Operations

Lemma 6.4.10

Given a junction tree, form a new cluster tree as follows. For each cluster C in the JT, choose an order of nodes within C, say $c_1, c_2, \ldots, c_k$, and hang a chain of clusters off of C consisting of $C \setminus \{c_1\}$ hanging from C, $C \setminus \{c_1, c_2\}$ hanging from $C \setminus \{c_1\}$, $C \setminus \{c_1, c_2, c_3\}$ hanging from $C \setminus \{c_1, c_2\}$, and so on. Then the resulting cluster graph is a cluster tree, and moreover it is still junction tree.

Lemma 6.4.11

Given a junction tree, where (C_i, C_j) are neighboring clusters in the tree, we can merge these two clusters forming a new cluster $C_{ij} = C_i \cup C_j$, and where the neighbors of C_{ij} are the set of neighbors of either C_i and C_j. Then the resulting structure is still junction tree.

If we keep doing the latter, we'll end up with one complete graph.

Key theorem: JT of maxcliques $\equiv$ triangulated graphs

Theorem 6.4.12

A graph $G = (V, E)$ is decomposable iff a junction tree of maxcliques for G exists.

Proof.

Induction on the number of maxcliques. If G has one maxclique, it is both a junction tree and decomposable. Assume true for $\leq k$ maxcliques and show it for $k + 1$.

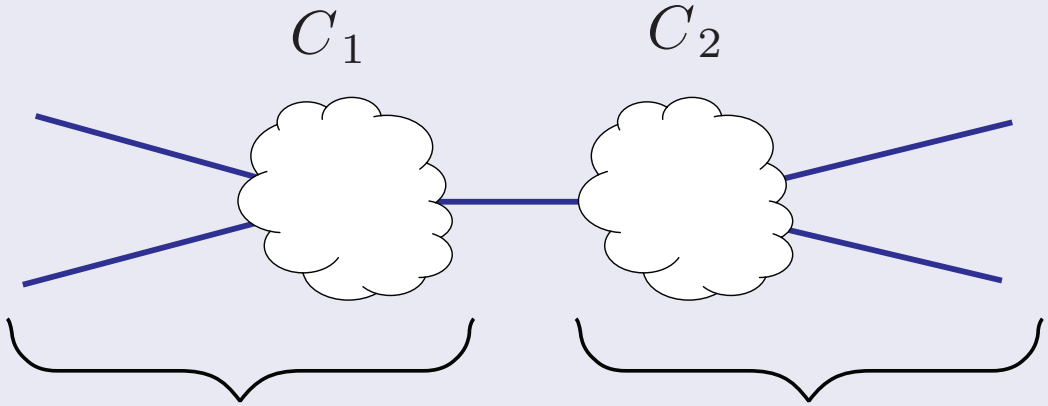
...

Junction tree of maxcliques $\equiv$ triangulated graphs

JT implies Decomposable

... proof continued.

a junction tree exists $\Rightarrow$ *decomposable:* Let $\mathcal{T}$ be a junction tree of maxcliques $\mathcal{C}$, and let C_1, C_2 be adjacent in $\mathcal{T}$. The edge C_1, C_2 in the tree separates $\mathcal{T}$ into two sub-trees $\mathcal{T}_1$ and $\mathcal{T}_2$, with V_i being the nodes in $\mathcal{T}_i$, $G_i = G[V_i]$ being the subgraph of G corresponding to $\mathcal{T}_i$, and $\mathcal{C}_i$ being the set of maxcliques in $\mathcal{T}_i$, for $i = 1, 2$. Thus $V(G) = V_1 \cup V_2$, and $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. Note that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. We also let $S = V_1 \cap V_2$ which is the intersection of all the nodes in each of the two trees.



Tree $\mathcal{T}_1$ with nodes V_1 forming graph $G_1 = G[V_1]$ and maxcliques $\mathcal{C}_1$.

Tree $\mathcal{T}_2$ with nodes V_2 forming graph $G_2 = G[V_2]$ and maxcliques $\mathcal{C}_2$.

Junction tree of maxcliques $\equiv$ triangulated graphs

JT implies Decomposable

... proof continued.

Also, the nodes in $\mathcal{T}_i$ are maxcliques in G_i and $\mathcal{T}_i$ is a junction tree for G_i since r.i.p. still holds in the subtrees of a junction tree. Therefore, by induction, G_i is decomposable. To show that G is decomposable, we need to show that: 1) $S = V_1 \cap V_2$ is complete, and 2) that S separates $G[V_1 \setminus S]$ from $G[V_2 \setminus S]$.

If $v \in S$, then for each G_i $(i = 1, 2)$, there exists a clique C_i' with $v \in C_i'$, and the path in $\mathcal{T}$ joining C_1' and C_2' passes through both C_1 and C_2. Because of the r.i.p., we thus have that $v \in C_1$ and $v \in C_2$ and so $v \in C_1 \cap C_2$. This means that $V_1 \cap V_2 \subseteq C_1 \cap C_2$. But $C_i \subseteq V_i$ since C_i is a clique in the corresponding tree $\mathcal{T}_i$. Therefore $C_1 \cap C_2 \subseteq V_1 \cap V_2 = S$, so that $S = C_1 \cap C_2$. This means that S contains all nodes that are common among the two subgraphs and moreover that S is complete as desired.

...

Junction tree of maxcliques $\equiv$ triangulated graphs

JT implies Decomposable

... proof continued.

Next, to show that S is a separator, we take $u \in V_1 \setminus S$ and $v \in V_2 \setminus S$ (note that such choices mean $u \notin V_2$ and $v \notin V_1$ due to the commonality property of S). Suppose the contrary that S does not separate V_1 from V_2, which means there exists a path $u, w_1, w_2, \ldots, w_k, v$ for the given u, v with $w_i \notin S$ for all i. Therefore, there is a clique $C \in \mathcal{C}$ containing the set $\{u, w_1\}$. We must have $C \notin \mathcal{C}_2$ since $u \notin V_2$, which means $C \in \mathcal{C}_1$ or $C \subseteq V_1$ implying that $w_1 \in V_1$ and moreover that $w_1 \in V_1 \setminus S$. We repeat this argument with w_1 taking the place of u and w_2 taking the place of w_1 in the path, and so on until we end up with $v \in V_1 \setminus S$ which is a contradiction. Therefore, S must separate V_1 from V_2. We have thus formed a decomposition of G as $(V_1 \setminus S, S, V_2 \setminus S)$ and since G_i is decomposable (by induction), we have that G is decomposable.

...

Junction tree of maxcliques $\equiv$ triangulated graphs

Decomposable implies JT

... proof continued.

decomposable $\Rightarrow$ *a junction tree exists:* Since G is decomposable, let (W_1, W_2, S) be a proper decomposition of G into decomposable subsets $G_1 = G[V_1]$ and $G_2 = G[V_2]$ with $V_i = W_i \cup S$. By induction, since G_1 and G_2 are decomposable, there exits a junction tree $\mathcal{T}_1$ and $\mathcal{T}_2$ corresponding to maxcliques in G_1 and G_2. Since this is a decomposition, with separator S, we can form all maxcliques $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ with $\mathcal{C}_i$ maxcliques of V_i for tree $\mathcal{T}_i$. Choose $C_1 \in \mathcal{C}_1$ and $C_2 \in \mathcal{C}_2$ such that $S \subseteq C_1$ and $S \subseteq C_2$ which is possible since S is complete, and must be contained in some maxclique in both $\mathcal{T}_1$ and $\mathcal{T}_2$. We form a new tree $\mathcal{T}$ by linking $C_1 \in \mathcal{T}_1$ with $C_2 \in \mathcal{T}_2$. We need next to ensure that this new junction tree satisfies r.i.p.

...

Junction tree of maxcliques $\equiv$ triangulated graphs

Decomposable implies JT

... proof continued.

Let $v \in V$. If $v \notin V_2$, then all cliques containing v are in $\mathcal{C}_1$ and those cliques form a connected tree by the junction tree property since $\mathcal{T}_1$ is a junction tree. The same is true if $v \notin V_1$. Otherwise, if $v \in S$ (meaning that $v \in V_1 \cap V_2$), then the cliques in $\mathcal{C}_i$ containing v are connected in $\mathcal{T}_i$ including C_i for $i = 1, 2$. But by forming $\mathcal{T}$ by connecting C_1 andC_2, and since v is arbitrary, we have retained the junction tree property. Thus, $\mathcal{T}$ is a junction tree.

□

Cliques or Maxcliques

Lemma 6.4.13

A junction tree of maxcliques for graph $G = (V, E)$ exists iff a junction tree of cliques for graph $G = (V, E)$ exists.

Cliques or Maxcliques

Lemma 6.4.13

A junction tree of maxcliques for graph $G = (V, E)$ exists iff a junction tree of cliques for graph $G = (V, E)$ exists.

- How can we get from one to the other?

Cliques or Maxcliques

Lemma 6.4.13

A junction tree of maxcliques for graph $G = (V, E)$ exists iff a junction tree of cliques for graph $G = (V, E)$ exists.

- How can we get from one to the other?

Since decomposable is same as triangulated:

Corollary 6.4.14

A graph G is triangulated iff a junction tree of cliques for G exists.

How to build a junction tree

- Maximum cardinality search algorithm can do this. If graph is triangulated, it produces a list of cliques in r.i.p. order.

Maximum Cardinality Search with maxclique order

Algorithm 2: Maximum Cardinality Search: Determines if a graph G is triangulated.

Input: An undirected graph $G = (V, E)$ with $n = |V|$.

Result: is triangulated?, if so MCS ordering $\sigma = (v_1, \ldots, v_n)$, and maxcliques in r.i.p. order.

$L \leftarrow \emptyset$; $i \leftarrow 1$; $\mathcal{C} \leftarrow \emptyset$;

while $|V \setminus L| > 0$ **do**

  Choose $v_i \in \text{argmax}_{u \in V \setminus L} |\delta(u) \cap L|$; /* v_i's previously labeled neighbors has max cardinality. */

  $c_i \leftarrow \delta(v_i) \cap L$; /* c_i is v_i's neighbors in the reverse elimination order. */

  if $\{v_i\} \cup c_i$ *is not complete in* G **then**

    return *"not triangulated"* ;

  if $|c_i| \leq |c_{i-1}|$ **then**

    $\mathcal{C} \leftarrow (\mathcal{C}, \{c_{i-1} \cup \{v_{i-1}\}\})$; /* Append the next maxclique to list $\mathcal{C}$. */

  if $i = n$ **then**

    $\mathcal{C} \leftarrow (\mathcal{C}, \{c_i \cup \{v_i\}\})$; ; /* Append the last maxclique to list $\mathcal{C}$. */

  $L \leftarrow L \cup \{v_i\}$ $i \leftarrow i + 1$;

return *"triangulated", the ordering* σ*, and the set of maxcliques* $\mathcal{C}$ *which are in r.i.p. order.*

Maximum Cardinality Search with maxclique order

Algorithm 3: Maximum Cardinality Search: Determines if a graph G is triangulated.

Input: An undirected graph $G = (V, E)$ with $n = |V|$.

Result: is triangulated?, if so MCS ordering $\sigma = (v_1, \ldots, v_n)$, and maxcliques in r.i.p. order.

$L \leftarrow \emptyset$; $i \leftarrow 1$; $\mathcal{C} \leftarrow \emptyset$;

while $|V \setminus L| > 0$ **do**

  Choose $v_i \in \text{argmax}_{u \in V \setminus L} |\delta(u) \cap L|$; /* v_i's previously labeled neighbors has max cardinality. */

  $c_i \leftarrow \delta(v_i) \cap L$; /* c_i is v_i's neighbors in the reverse elimination order. */

  if $\{v_i\} \cup c_i$ *is not complete in* G **then**

    return *"not triangulated"* ;

  if $|c_i| \leq |c_{i-1}|$ **then**

    $\mathcal{C} \leftarrow (\mathcal{C}, \{c_{i-1} \cup \{v_{i-1}\}\})$; /* Append the next maxclique to list $\mathcal{C}$. */

  if $i = n$ **then**

    $\mathcal{C} \leftarrow (\mathcal{C}, \{c_i \cup \{v_i\}\})$; ; /* Append the last maxclique to list $\mathcal{C}$. */

  $L \leftarrow L \cup \{v_i\}$ $i \leftarrow i + 1$;

return *"triangulated", the ordering* σ*, and the set of maxcliques* $\mathcal{C}$ *which are in r.i.p. order.*

How to build a junction tree

- Alternatively, we can construct the maxcliques in any form (say by running elimination) and find a maximal spanning tree over the edge-weighted cluster graph, where clusters correspond to maxcliques, and edge weights correspond to the size of the intersection of the two adjacent maxcliques.

How to build a junction tree

- Alternatively, we can construct the maxcliques in any form (say by running elimination) and find a maximal spanning tree over the edge-weighted cluster graph, where clusters correspond to maxcliques, and edge weights correspond to the size of the intersection of the two adjacent maxcliques.
- Prim's algorithm can run in $O(|E| + |V| \log |V|)$, much better than $|V|^2$ for sparse graphs.

How to build a junction tree

- Alternatively, we can construct the maxcliques in any form (say by running elimination) and find a maximal spanning tree over the edge-weighted cluster graph, where clusters correspond to maxcliques, and edge weights correspond to the size of the intersection of the two adjacent maxcliques.
- Prim's algorithm can run in $O(|E| + |V| \log |V|)$, much better than $|V|^2$ for sparse graphs.

Theorem 6.4.15

A tree of maxcliques $\mathcal{T}$ is a junction tree iff it is a maximum spanning tree on the maxclique graph, with edge weights set according to the cardinality of the separator between the two maxcliques.

How to build a junction tree

- Alternatively, we can construct the maxcliques in any form (say by running elimination) and find a maximal spanning tree over the edge-weighted cluster graph, where clusters correspond to maxcliques, and edge weights correspond to the size of the intersection of the two adjacent maxcliques.
- Prim's algorithm can run in $O(|E| + |V| \log |V|)$, much better than $|V|^2$ for sparse graphs.

Theorem 6.4.15

A tree of maxcliques $\mathcal{T}$ is a junction tree iff it is a maximum spanning tree on the maxclique graph, with edge weights set according to the cardinality of the separator between the two maxcliques.

- Note: graph must be triangulated. I.e., maximum spanning tree of a cluster graph where the clusters are maxcliques but the graph is not triangulated will clearly not produce a junction tree.

Other aspects of JTs

- There can be multiple JTs for a given triangulated graph (e.g., consider any graph where $d(S) \geq 3$ for some separator S).

Other aspects of JTs

- There can be multiple JTs for a given triangulated graph (e.g., consider any graph where $d(S) \geq 3$ for some separator S).
- JTs are not binary decomposition trees (BDTs), but they are related. Leaf nodes of BDTs correspond to nodes in a JT of maxcliques. Non-leaf nodes in a BDTs may correspond to edges in a JT. Therefore, edges in a JT may correspond to all minimal separators in triangulated graph G'.

Other aspects of JTs

- There can be multiple JTs for a given triangulated graph (e.g., consider any graph where $d(S) \geq 3$ for some separator S).
- JTs are not binary decomposition trees (BDTs), but they are related. Leaf nodes of BDTs correspond to nodes in a JT of maxcliques. Non-leaf nodes in a BDTs may correspond to edges in a JT. Therefore, edges in a JT may correspond to all minimal separators in triangulated graph G'.
- Set of maxcliques is unique in a triangulated graph. Set of minimal separators is unique in a triangulated graph.

Other aspects of JTs

- There can be multiple JTs for a given triangulated graph (e.g., consider any graph where $d(S) \geq 3$ for some separator S).
- JTs are not binary decomposition trees (BDTs), but they are related. Leaf nodes of BDTs correspond to nodes in a JT of maxcliques. Non-leaf nodes in a BDTs may correspond to edges in a JT. Therefore, edges in a JT may correspond to all minimal separators in triangulated graph G'.
- Set of maxcliques is unique in a triangulated graph. Set of minimal separators is unique in a triangulated graph.
- Again, JT can be over not just maxcliques. JT can exist over all cliques, or over some cliques (if they contain all maxcliques)

Other aspects of JTs

- There can be multiple JTs for a given triangulated graph (e.g., consider any graph where $d(S) \geq 3$ for some separator S).
- JTs are not binary decomposition trees (BDTs), but they are related. Leaf nodes of BDTs correspond to nodes in a JT of maxcliques. Non-leaf nodes in a BDTs may correspond to edges in a JT. Therefore, edges in a JT may correspond to all minimal separators in triangulated graph G'.
- Set of maxcliques is unique in a triangulated graph. Set of minimal separators is unique in a triangulated graph.
- Again, JT can be over not just maxcliques. JT can exist over all cliques, or over some cliques (if they contain all maxcliques)
- Different JTs always have same set of nodes and separators, just different configurations.

Intersection Graphs

- We're next going to look at seemingly very different way to view triangulated graphs and junction trees, based on intersection graph theory.

Intersection Graphs

- We're next going to look at seemingly very different way to view triangulated graphs and junction trees, based on intersection graph theory.
- We'll see that triangulated graphs are identical to a type of intersection graph, where the underlying object is a tree (furthering our connection to trees).

Intersection Graphs

- We're next going to look at seemingly very different way to view triangulated graphs and junction trees, based on intersection graph theory.
- We'll see that triangulated graphs are identical to a type of intersection graph, where the underlying object is a tree (furthering our connection to trees).
- first, lets talk a bit about terminology.

Edge Clique Covers

- Set cover - sets must cover the ground/universal set (ground set cover)

Edge Clique Covers

- Set cover - sets must cover the ground/universal set (ground set cover)
- Vertex cover - vertices must cover the edges (edge vertex cover)

Edge Clique Covers

- Set cover - sets must cover the ground/universal set (ground set cover)
- Vertex cover - vertices must cover the edges (edge vertex cover)
- Edge cover - edges must cover the vertices (vertex edge cover)

Edge Clique Covers

- Set cover - sets must cover the ground/universal set (ground set cover)
- Vertex cover - vertices must cover the edges (edge vertex cover)
- Edge cover - edges must cover the vertices (vertex edge cover)
- clique cover - cliques cover the edges (edge clique cover)

Edge Clique Covers

- Set cover - sets must cover the ground/universal set (ground set cover)
- Vertex cover - vertices must cover the edges (edge vertex cover)
- Edge cover - edges must cover the vertices (vertex edge cover)
- clique cover - cliques cover the edges (edge clique cover)
- The nodes of a junction tree of cliques (or maxcliques) constitute an edge clique cover for triangulated graph G' — start with set of nodes $V = \cup_{C \in \mathcal{C}} C$. Add edge between $u, v \in V$ if exists a $C \in \mathcal{C}$ such that $u, v \in C$.

Edge Clique Covers

- Set cover - sets must cover the ground/universal set (ground set cover)
- Vertex cover - vertices must cover the edges (edge vertex cover)
- Edge cover - edges must cover the vertices (vertex edge cover)
- clique cover - cliques cover the edges (edge clique cover)
- The nodes of a junction tree of cliques (or maxcliques) constitute an edge clique cover for triangulated graph G' — start with set of nodes $V = \cup_{C \in \mathcal{C}} C$. Add edge between $u, v \in V$ if exists a $C \in \mathcal{C}$ such that $u, v \in C$.
- Going from G' to JT and back to the graph yields the same graph.

Intersection Graphs

Definition 6.5.1 (Intersection Graph)

An intersection graph is a graph $G = (V, E)$ where each vertex $v \in V(G)$ corresponds to a set U_v and each edge $(u, v) \in E(G)$ exists only if $U_u \cap U_v \neq \emptyset$.

Intersection Graphs

Definition 6.5.1 (Intersection Graph)

An intersection graph is a graph $G = (V, E)$ where each vertex $v \in V(G)$ corresponds to a set U_v and each edge $(u, v) \in E(G)$ exists only if $U_u \cap U_v \neq \emptyset$.

- some underlying set of objects U and a **multiset** of subsets of U of the form $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$ with $U_i \subseteq U$ — might have some i, j where $U_i = U_j$.

Intersection Graphs

Definition 6.5.1 (Intersection Graph)

An intersection graph is a graph $G = (V, E)$ where each vertex $v \in V(G)$ corresponds to a set U_v and each edge $(u, v) \in E(G)$ exists only if $U_u \cap U_v \neq \emptyset$.

- some underlying set of objects U and a **multiset** of subsets of U of the form $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$ with $U_i \subseteq U$ — might have some i, j where $U_i = U_j$.

Theorem 6.5.2

Every graph is an intersection graph.

Intersection Graphs

Definition 6.5.1 (Intersection Graph)

An intersection graph is a graph $G = (V, E)$ where each vertex $v \in V(G)$ corresponds to a set U_v and each edge $(u, v) \in E(G)$ exists only if $U_u \cap U_v \neq \emptyset$.

- some underlying set of objects U and a **multiset** of subsets of U of the form $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$ with $U_i \subseteq U$ — might have some i, j where $U_i = U_j$.

Theorem 6.5.2

Every graph is an intersection graph.

This can be seen informally by consider an arbitrary graph, create a U_i for every node, and construct the subsets so that the edges will exist when taking intersection.

Interval Graphs (a type of intersetion graph)

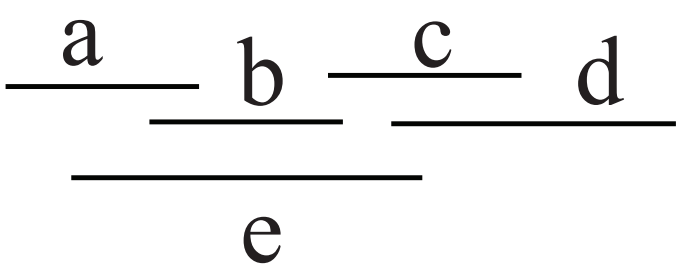
- Interval graphs are intersection graphs where the subsets are intervals/segments $[a, b]$ in $\mathbb{R}$

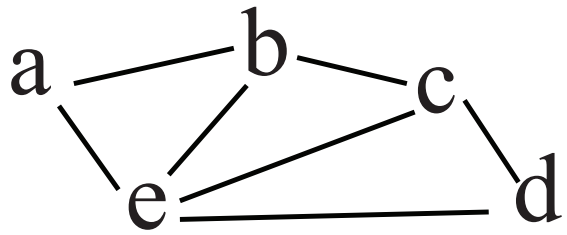
Interval Graphs (a type of intersetion graph)

- Interval graphs are intersection graphs where the subsets are intervals/segments $[a, b]$ in $\mathbb{R}$
- Any graph that can be constructed this way is an interval graph

Interval Graphs (a type of intersetion graph)

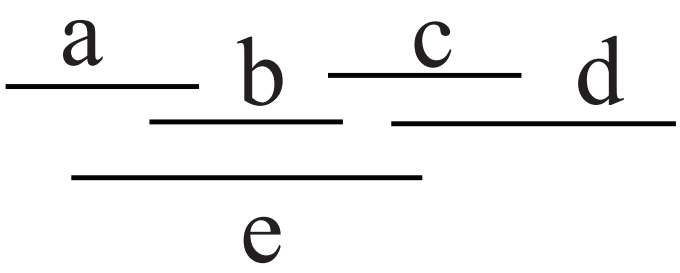
- Interval graphs are intersection graphs where the subsets are intervals/segments $[a, b]$ in $\mathbb{R}$
- Any graph that can be constructed this way is an interval graph

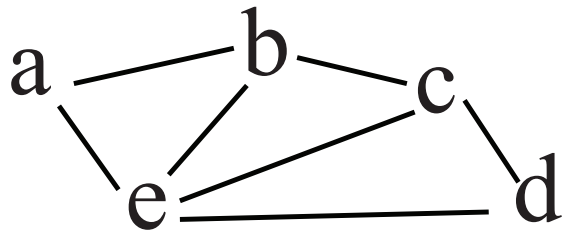




Interval Graphs (a type of intersetion graph)

- Interval graphs are intersection graphs where the subsets are intervals/segments $[a, b]$ in $\mathbb{R}$
- Any graph that can be constructed this way is an interval graph

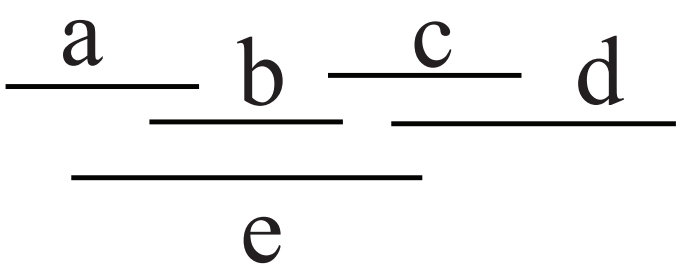


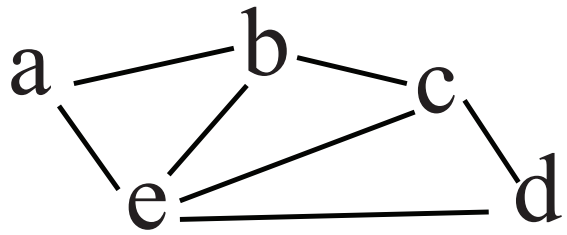


- Are all graphs interval graphs?

Interval Graphs (a type of intersetion graph)

- Interval graphs are intersection graphs where the subsets are intervals/segments $[a, b]$ in $\mathbb{R}$
- Any graph that can be constructed this way is an interval graph





- Are all graphs interval graphs? 4-cycle

Interval Graphs

Theorem 6.5.3

All Interval Graphs are triangulated.

proof sketch.

Given interval graph $G = (V, E)$, consider any cycle $u, w_1, w_2, \ldots, w_k, v, u \in V(G)$. Cycle must go (w.l.o.g.) forward and then backwards along the line in order to connect back to u, so there must be a chord between some non-adjacent nodes (since they will overlap). □

Are all triangulated graphs interval graphs?

Interval Graphs

Theorem 6.5.3

All Interval Graphs are triangulated.

proof sketch.

Given interval graph $G = (V, E)$, consider any cycle $u, w_1, w_2, \ldots, w_k, v, u \in V(G)$. Cycle must go (w.l.o.g.) forward and then backwards along the line in order to connect back to u, so there must be a chord between some non-adjacent nodes (since they will overlap). □

Are all triangulated graphs interval graphs? No, consider spider graph (elongated star graph).

Sub-tree intersection Graphs

- Given underlying tree, create intersection graph, where subsets are (nec. connected) subtrees of some "ground" tree.

Sub-tree intersection Graphs

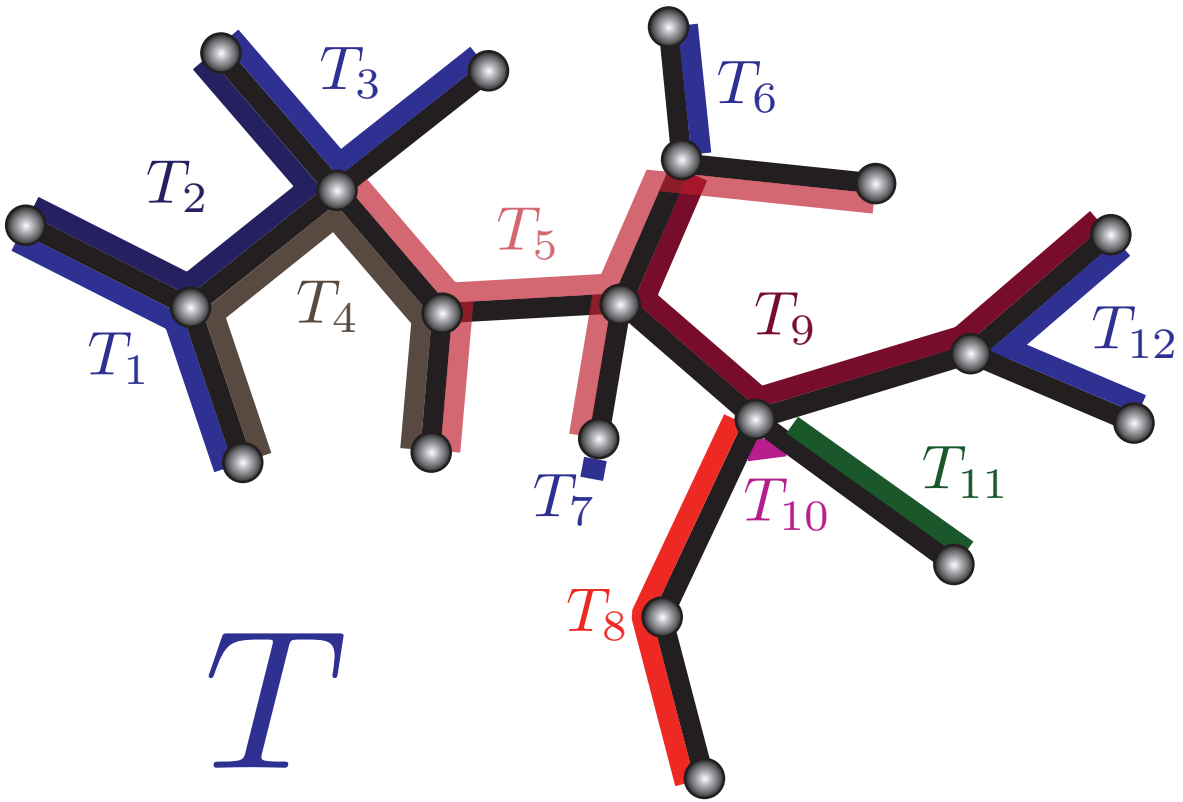
- Given underlying tree, create intersection graph, where subsets are (nec. connected) subtrees of some "ground" tree.
- Intersection exists if there are any nodes in common amongst the two corresponding trees.

Sub-tree intersection Graphs

- Given underlying tree, create intersection graph, where subsets are (nec. connected) subtrees of some "ground" tree.
- Intersection exists if there are any nodes in common amongst the two corresponding trees.

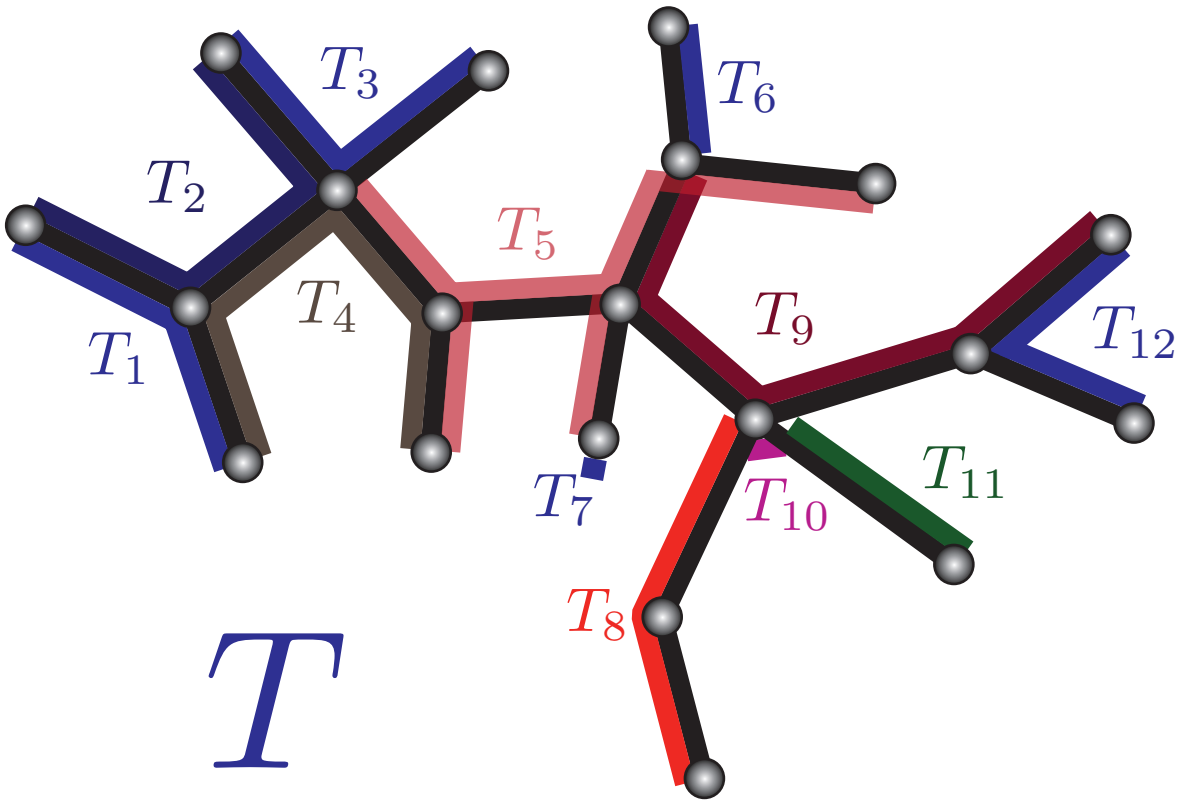
Sub-tree intersection Graphs

- Given underlying tree, create intersection graph, where subsets are (nec. connected) subtrees of some "ground" tree.
- Intersection exists if there are any nodes in common amongst the two corresponding trees.

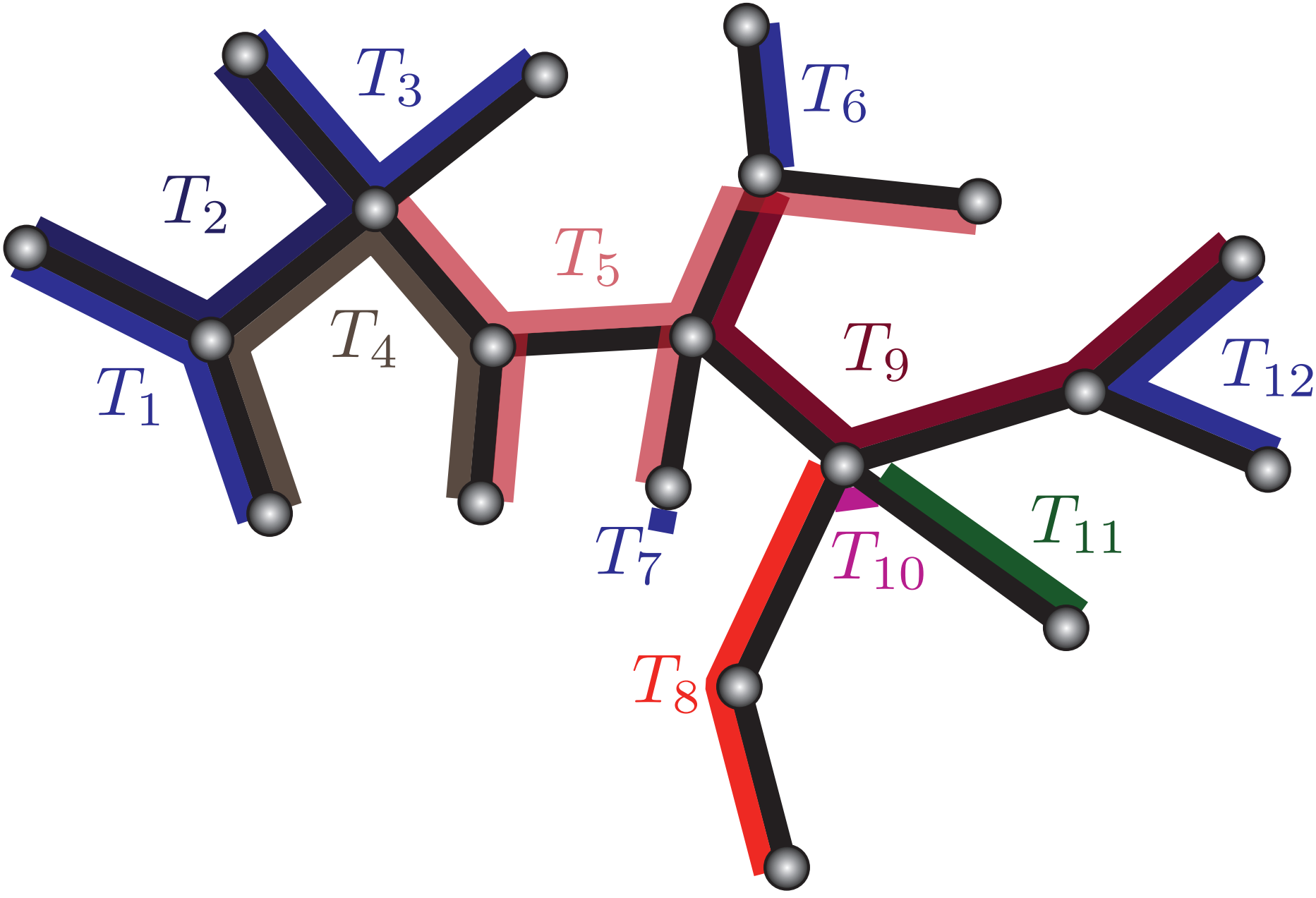


Sub-tree intersection Graphs

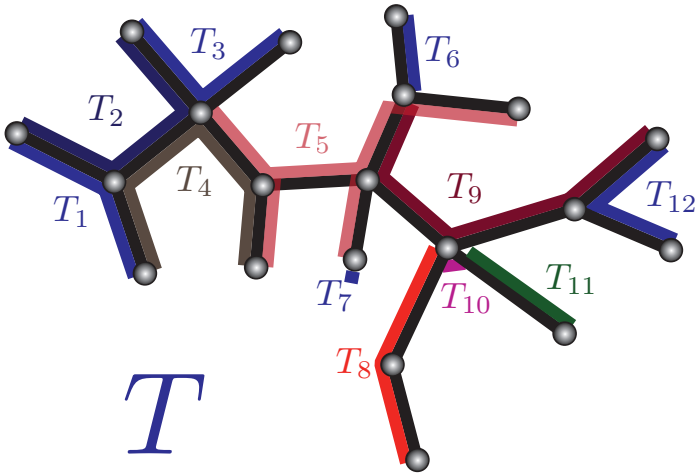
- Given underlying tree, create intersection graph, where subsets are (nec. connected) subtrees of some "ground" tree.
- Intersection exists if there are any nodes in common amongst the two corresponding trees.

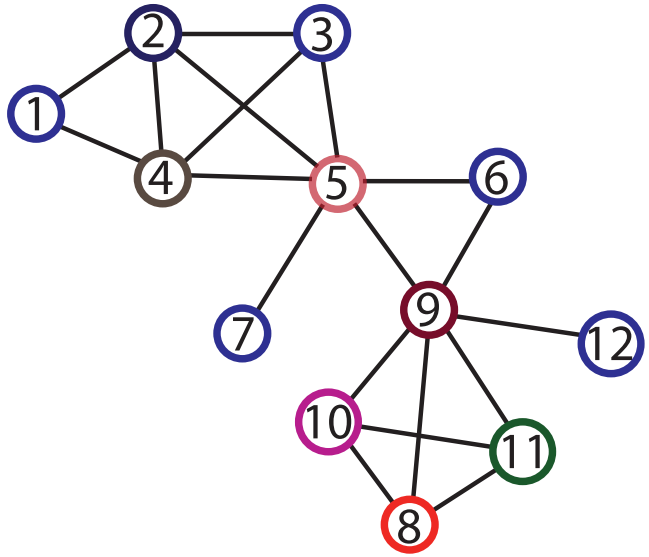


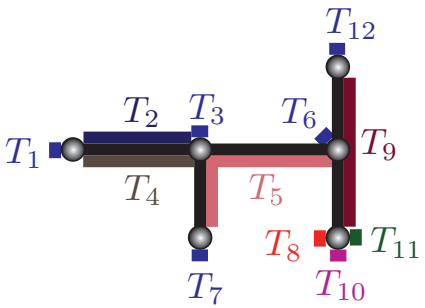
Lets zoom in a little on this


T_1
T_2
T_3
T_4
T_5
T_6
T_7
T_8
T_9
T_{10}
T_{11}
T_{12}

Sub-tree intersection Graphs

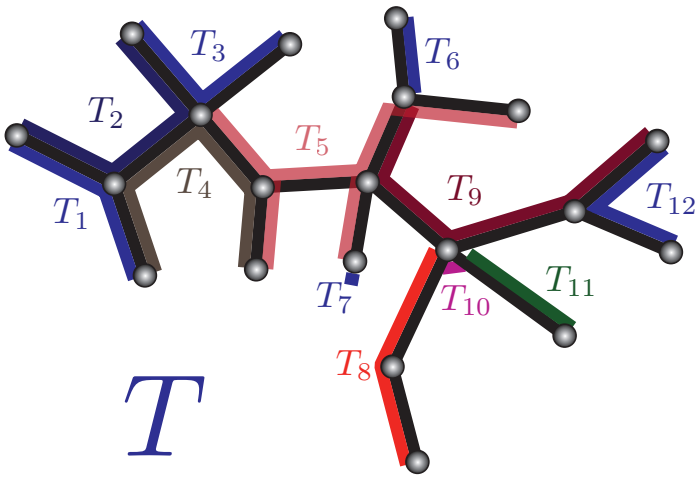


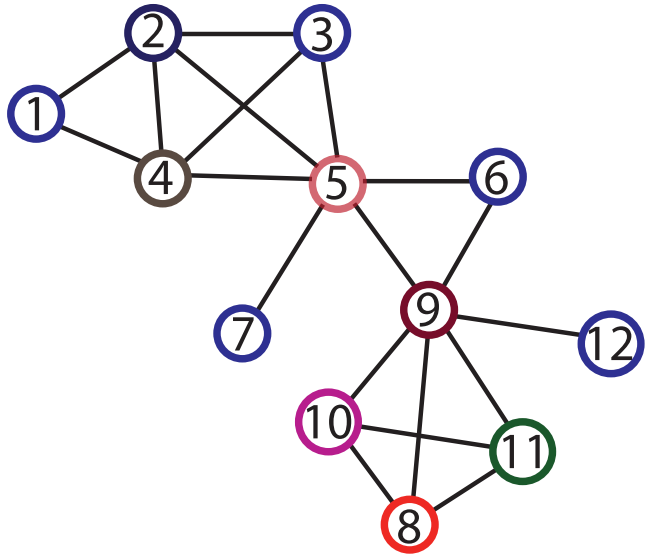


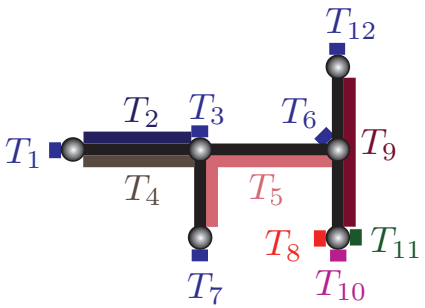


- Intersection exists if there are any nodes in common amongst the two corresponding trees.

Sub-tree intersection Graphs

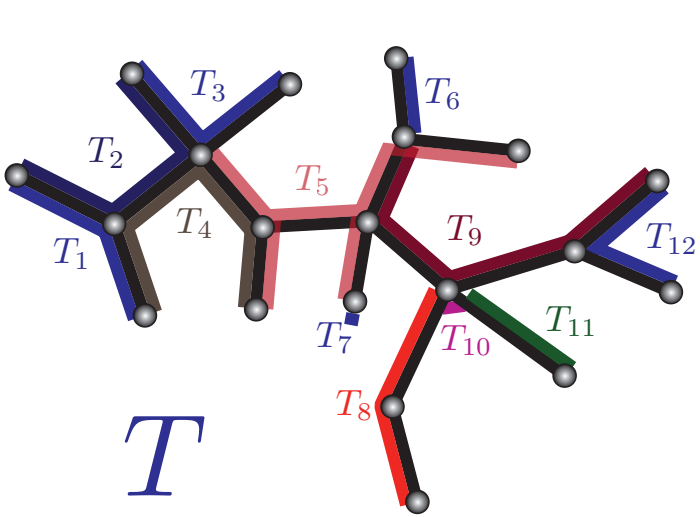


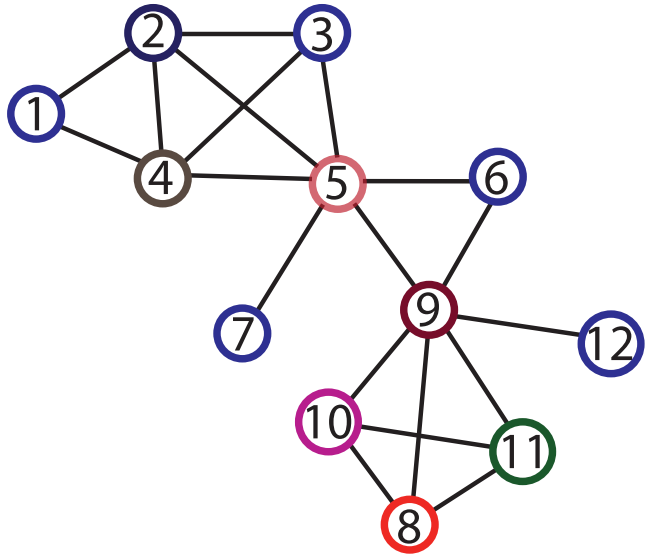


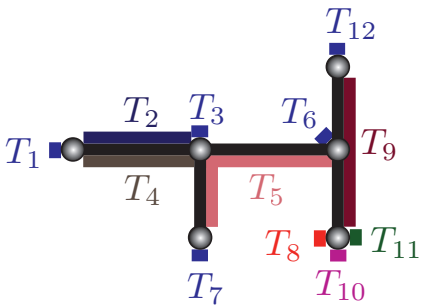


- Intersection exists if there are any nodes in common amongst the two corresponding trees.
- A sub-tree graph corresponds to more than one underlying tree (thus ground set and underlying subsets).

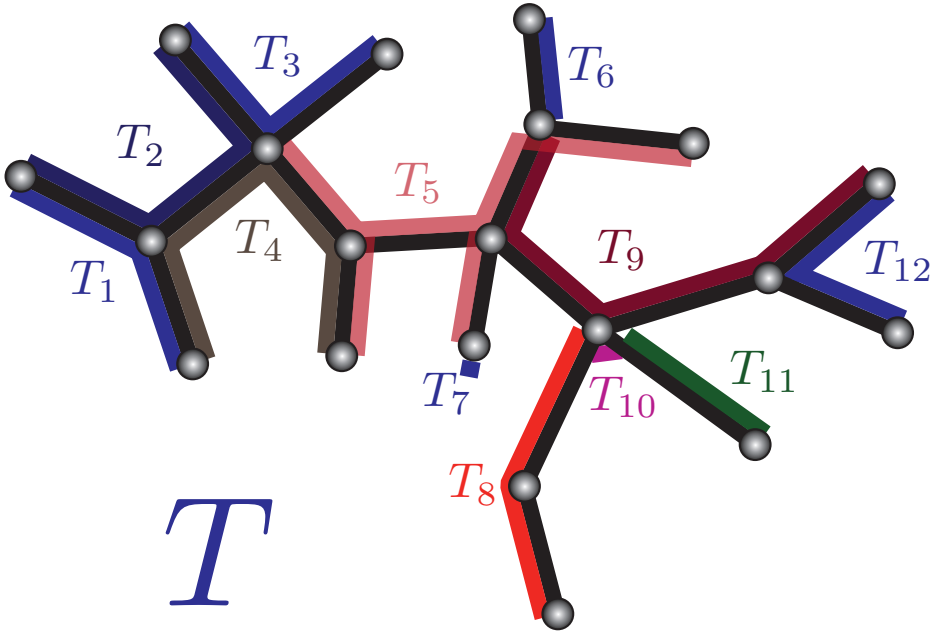
Sub-tree intersection Graphs

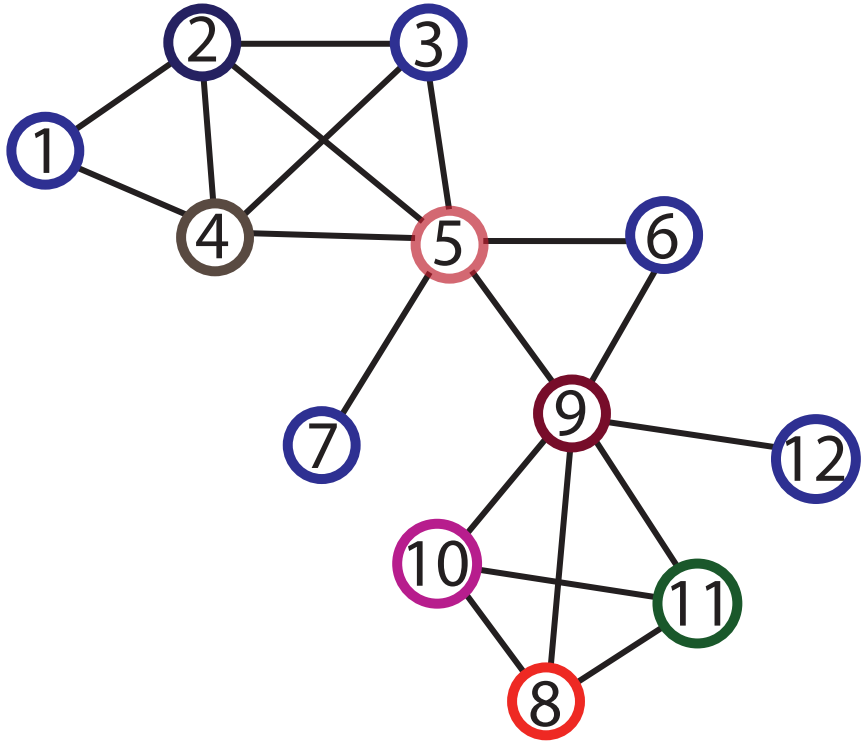


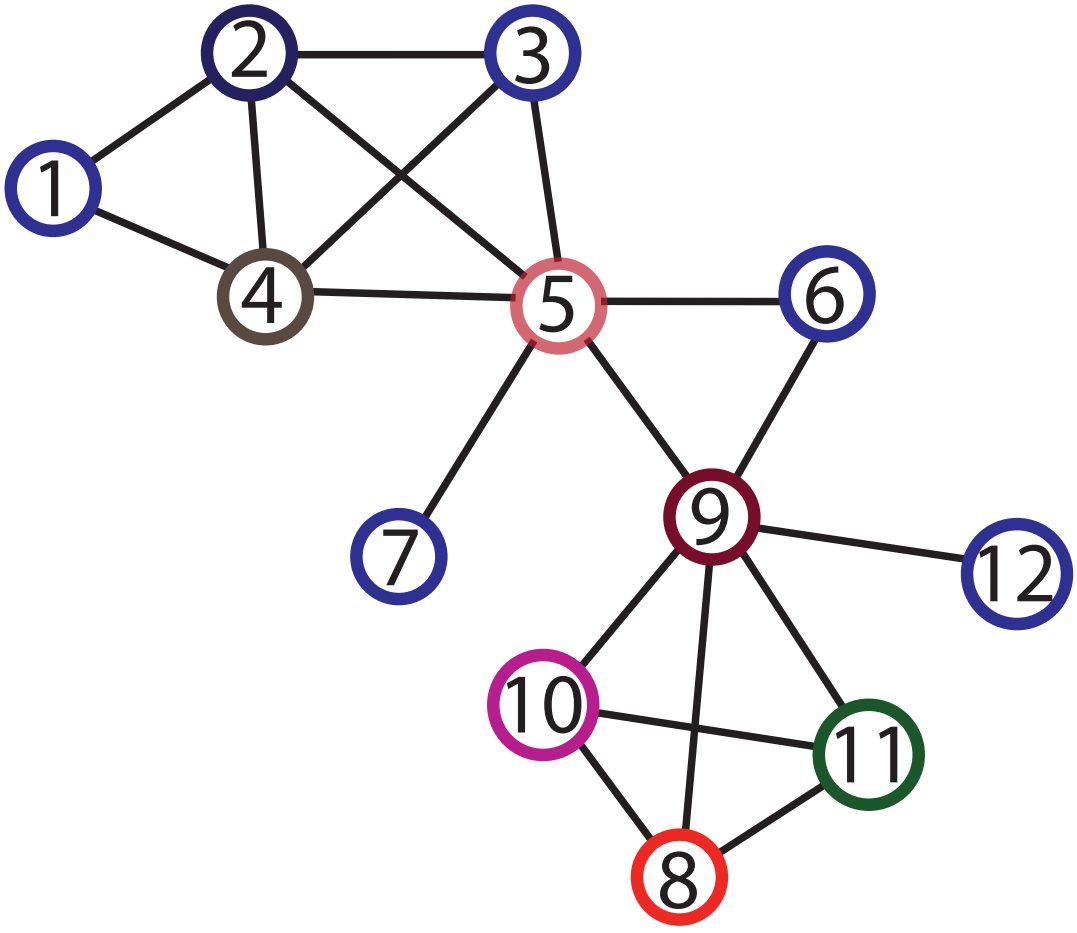


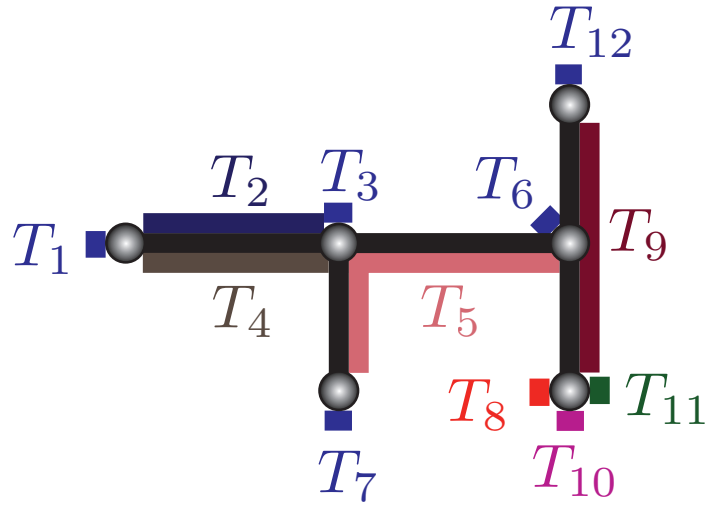


- Intersection exists if there are any nodes in common amongst the two corresponding trees.
- A sub-tree graph corresponds to more than one underlying tree (thus ground set and underlying subsets).
- What is the difference between left and right trees?

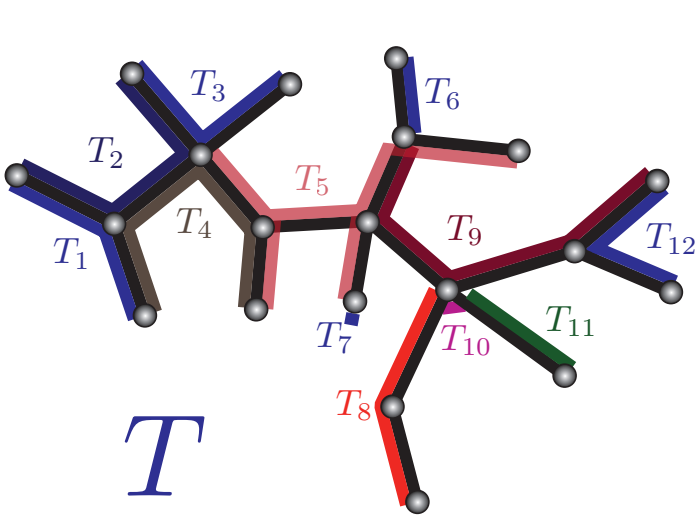

T_1
T_2
T_3
T_4
T_5
T_6
T_7
T_8
T_9
T_{10}
T_{11}
T_{12}
T

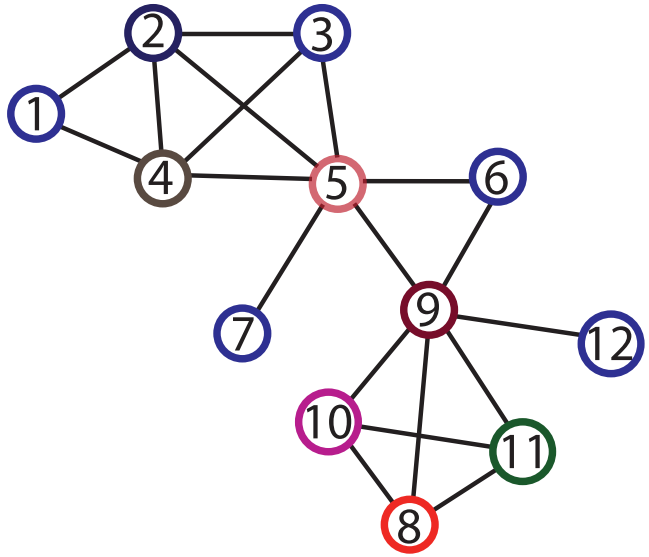

1
2
3
4
5
6
7
8
9
10
11
12

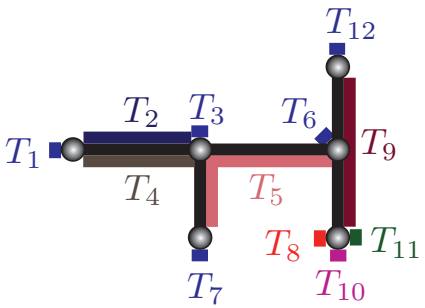

1
2
3
4
5
6
7
9
12
10
11
8


T_{12}
T_2
T_3
T_6
T_9
T_1
T_4
T_5
T_8
T_{11}
T_7
T_{10}

Sub-tree intersection Graphs

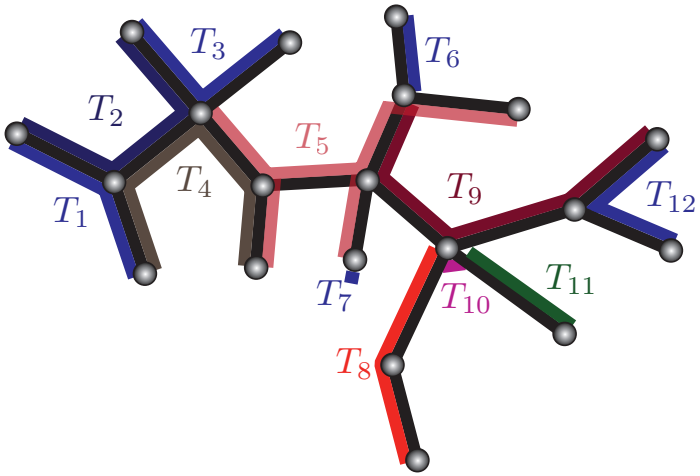


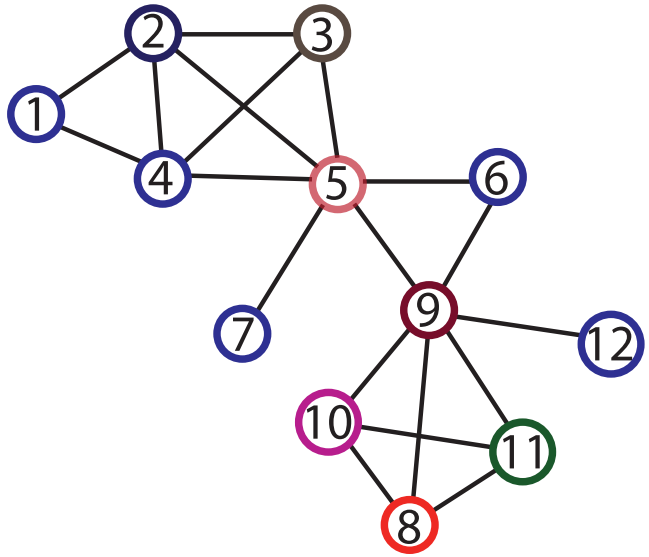


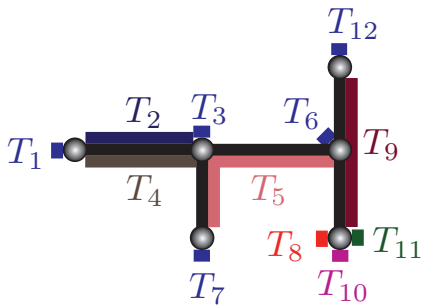


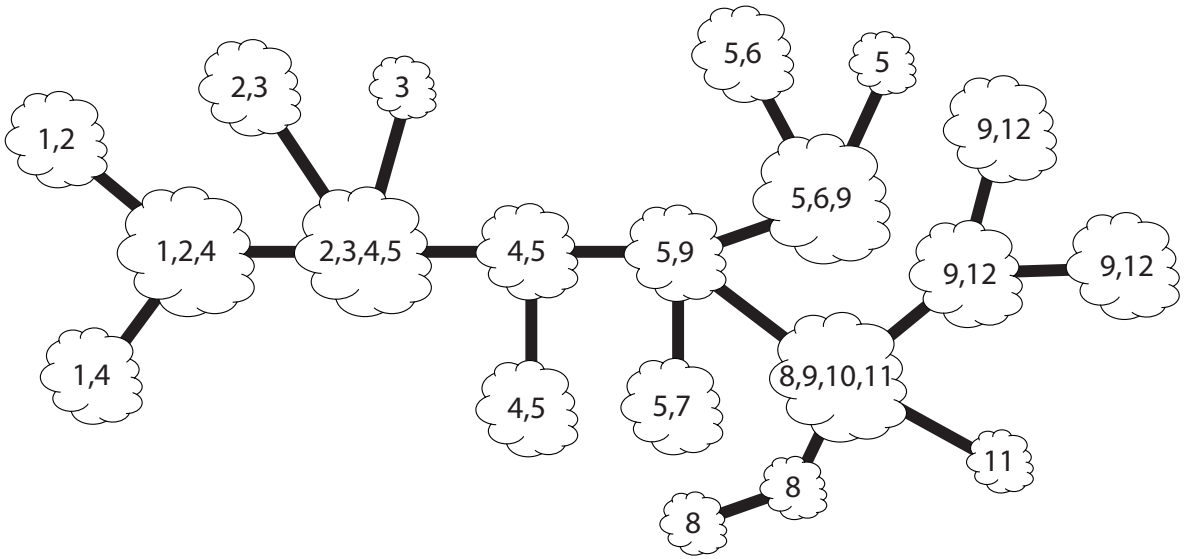
- Intersection exists if there are any nodes in common amongst the two corresponding trees.
- A sub-tree graph corresponds to more than one underlying tree (thus ground set and underlying subsets).
- What is the difference between left and right trees?
- Junction tree of cliques and maxcliques vs. junction tree of just maxcliques.

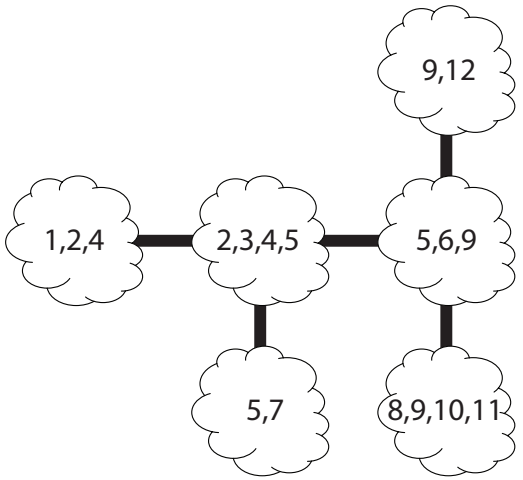
Sub-tree intersection Graphs w. Junction Trees











Sub-tree intersection graphs

Theorem 6.5.4

A graph $G = (V, E)$ is triangulated iff it corresponds to a sub-tree graph (i.e., an intersection graph on subtrees of some tree).

proof sketch.

We see that any sub-tree graph is such that nodes in the tree correspond to cliques in G, and by the nature of how the graph is constructed (subtrees of some underlying tree), the tree corresponds to a cluster tree that satisfies the induced subtree property. Therefore, any sub-tree graph corresponds to a junction tree, and any corresponding graph G is triangulated. □

Sub-tree intersection graphs

- All interval graphs are sub-tree intersection graphs (underlying tree is a chain, subtrees are sub-chains)

Sub-tree intersection graphs

- All interval graphs are sub-tree intersection graphs (underlying tree is a chain, subtrees are sub-chains)
- Are all sub-tree intersection graphs interval graphs?

Sub-tree intersection graphs

- All interval graphs are sub-tree intersection graphs (underlying tree is a chain, subtrees are sub-chains)
- Are all sub-tree intersection graphs interval graphs?
- So sub-tree intersection graphs capture the "tree-like" nature of triangulated graphs.

Sub-tree intersection graphs

- All interval graphs are sub-tree intersection graphs (underlying tree is a chain, subtrees are sub-chains)
- Are all sub-tree intersection graphs interval graphs?
- So sub-tree intersection graphs capture the "tree-like" nature of triangulated graphs.
- Triangulated graphs are also called hyper-trees (specific type of hyper-graph, where edges are generalized to be clusters of nodes rather than 2 nodes in a normal graph). In hyper-tree, the unique "max-edge" path between any two nodes property is generalized.

Sources for Today's Lecture

- Most of this material comes from the reading handout `tree_inference.pdf`