EE512A – Advanced Inference in Graphical Models

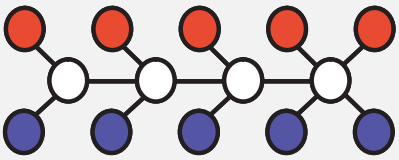
— Fall Quarter, Lecture 4 —

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 8th, 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.
- Slides from previous time this course was offered are at our previous web page (http://j.ee.washington.edu/~bilmes/classes/ee512a_fall_2011/) and even earlier at http://melodi.ee.washington.edu/~bilmes/ee512fa09/.

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, Triangulation, Multiple queries, Junction Trees
- L6 (10/15):
- 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.

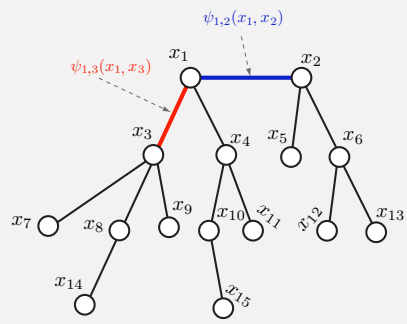
Generic form of message

$$\mu_{i\to j}(x_j) = \sum_{x_i} \left(\psi_{i,j}(x_i, x_j) \prod_{k\in\delta(i)\setminus\{j\}} \mu_{k\to i}(x_i) \right) \tag{4.5}$$

Message is of form:

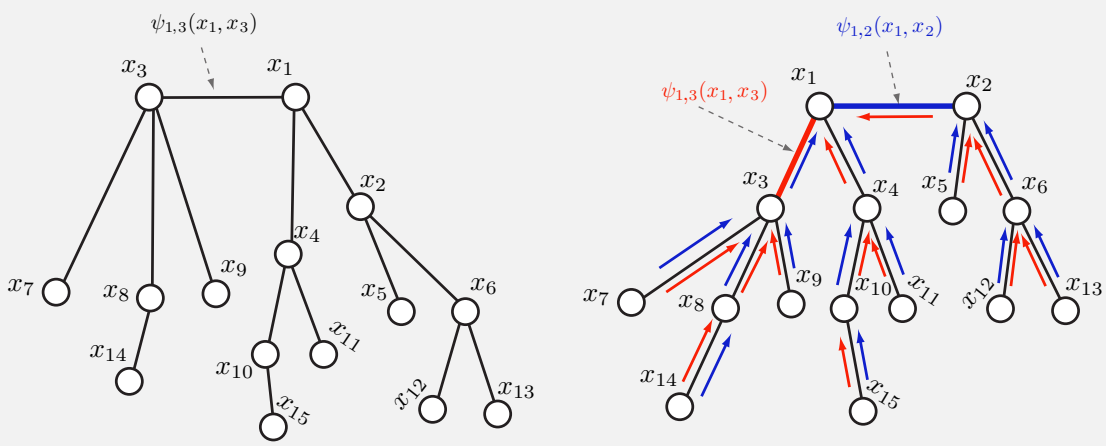
1. First, collect messages from all neighbors of i other than j,
2. next, incorporate these incoming messages by multiplying them in along with the factor $\psi_{i,j}(x_i, x_j)$,
3. the factor $\psi_{i,j}(x_i, x_j)$ relates x_i and x_j, and can be seen as a representation of a "communications channel" relating how the information x_i transforms into the information in x_j, thus motivating the terminology of a "message", and
4. then finally marginalizing away x_i thus yielding the desired message to be delivered at the destination node x_j.

Multiple Tree Queries: Variable elimination



- For $p(x_1, x_2)$, the variable elimination ordering $(14, 7, 8, 9, 15, 10, 11, 4, 12, 13, 5, 6, 3)$ would suffice
- 13 messages: $\mu_{14\to 8}(x_8)$, $\mu_{7\to 3}(x_3)$, $\mu_{8\to 3}(x_3)$, $\mu_{9\to 3}(x_3)$, $\mu_{15\to 10}(x_{10})$, $\mu_{10\to 4}(x_4)$, $\mu_{11\to 4}(x_4)$, $\mu_{4\to 1}(x_1)$, $\mu_{12\to 6}(x_6)$, $\mu_{13\to 6}(x_6)$, $\mu_{5\to 2}(x_2)$, $\mu_{6\to 2}(x_2)$, and $\mu_{3\to 1}(x_1)$.
- For $p(x_1, x_3)$, the variable ordering $(14, 7, 8, 9, 15, 10, 11, 4, 12, 13, 5, 6, 2)$ would suffice
- messages: $\mu_{14\to 8}(x_8)$, $\mu_{7\to 3}(x_3)$, $\mu_{8\to 3}(x_3)$, $\mu_{9\to 3}(x_3)$, $\mu_{15\to 10}(x_{10})$, $\mu_{10\to 4}(x_4)$, $\mu_{11\to 4}(x_4)$, $\mu_{4\to 1}(x_1)$, $\mu_{12\to 6}(x_6)$, $\mu_{13\to 6}(x_6)$, $\mu_{5\to 2}(x_2)$, $\mu_{6\to 2}(x_2)$, and $\mu_{2\to 1}(x_1)$.
- First 12 of variables in each order are identical! Results in marginal $p(x_1, x_2, x_3)$ from which both results are easy.

Multiple Tree Queries

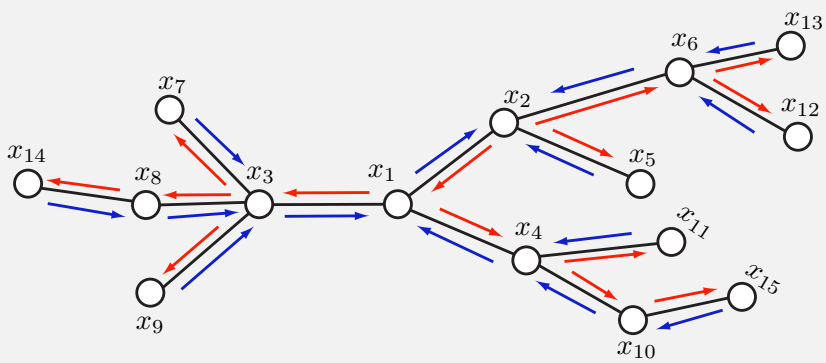


.

- Another look: Left tree rooted at $(1, 3)$, right rooted at $(1, 2)$.
- Red arrows are messages are for $(1, 3)$, blue arrows are messages for $(1, 2)$.
- most messages are the same.

All edge Queries

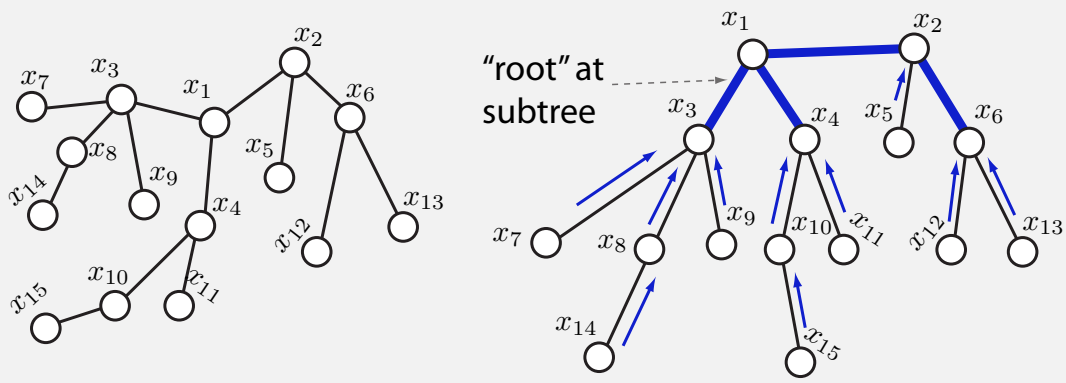
- As number of queries increases, so does efficiency (queries/message)
- Consider computing $p(x_i, x_j)$ for all $(i, j) \in E(G)$.
- Naive case, $N - 1$ edges $O(N^2 r^2)$.
- Smart case, only $O(Nr^2)$ still.
- consider: root tree at all $(i, j) \in E(G)$ in turn
- mark edge with arrow only once (so don't redundantly send message)
- result is each edge has two arrows in each direction



Collect/Distribute Evidence and MPP

- At the collect evidence stage, a message is not sent to a node's (single) parent until it has received messages from all its children, so there is only one node it has not yet received a message from, namely the parent.
- At the distribute evidence stage, once a node has received a message from its parent, it has received a message from all of its neighbors (since it received a message from all its children earlier, during the collect evidence phase) so it is free to send a message to any child that it likes.
- All messages obey the message passing protocol.
- Collect Evidence: a post-order tree traversal.
- Distribute Evidence: a pre-order tree traversal.

Tree queries with arbitrary S



- Above, $S = \{1, 2, 3, 4, 6\}$ which induces a sub-tree in G, so all messages sent towards nearest node inside of S.
- Once we have $p(x_S)$ we have efficient representation for it, using only r^2 tables.

Tree queries with arbitrary S

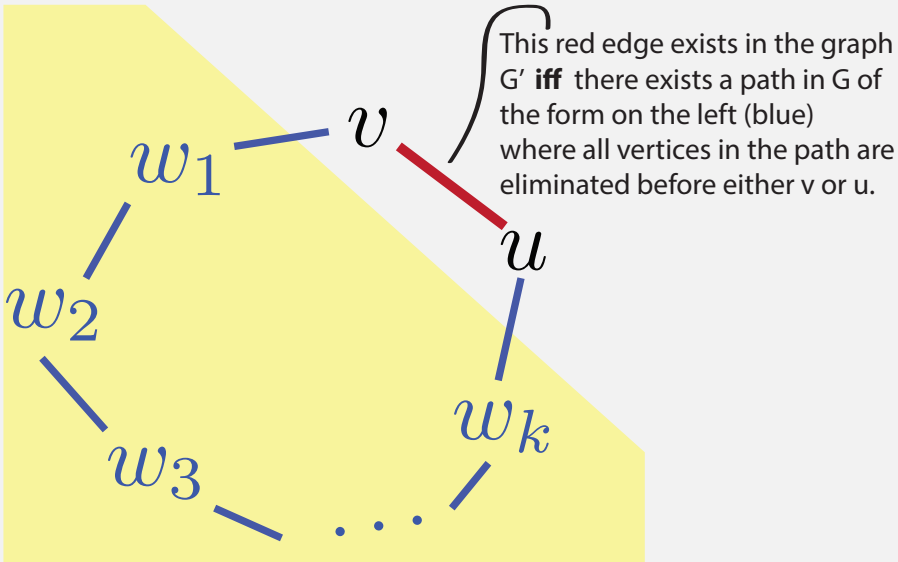
- We eliminate $x_{V \setminus S}$, which might introduce edges.
- Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ be an ordering of the nodes. Also $\sigma^{-1}(v)$ for $v \in V(G)$ gives number that node v is eliminated by order σ. We have following theorem

Theorem 4.2.2 (Rose's Entanglement Theorem (Lemma 4 of Rose 1976))

Let $G = (V, E)$ be an undirected graph with a given elimination ordering σ that maps G to $G' = (V, E')$ where $E' = E \cup F_\sigma$, and where F_σ are the fill-in edges added during elimination with order σ. Then $(v, w) \in E'$ is an edge in G' **iff** *there is a path in G with endpoints v and w, and where any nodes on the path other than v and w are eliminated before v and w in order σ. I.e., if there is a path $(v = v_1, v_2, \ldots, v_{k+1} = w)$ in G such that*

$$\sigma^{-1}(v_i) < \min(\sigma^{-1}(v), \sigma^{-1}(w)), \text{ for } 2 \leq i \leq k \tag{4.11}$$

Rose's theorem: figure



- If we eliminate all of $w_1, w_2, \ldots, w_k$ before we eliminate u and v, then we will necessarily have an edge between u and v.

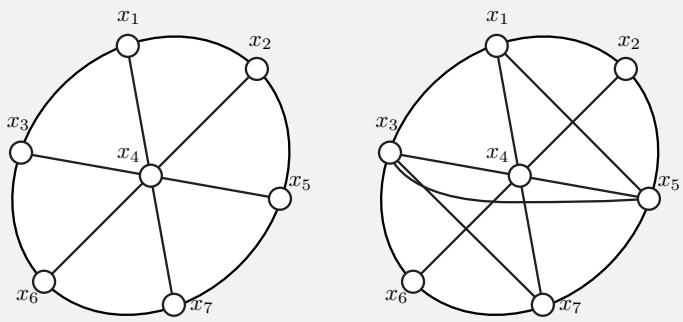
Perfect elimination orders

Definition 4.2.2 (perfect elimination order)

Order σ is called perfect for G if when we eliminate nodes in G according to σ, there are zero fill edges in the resulting reconstituted graph.

- For a tree, there is always a perfect elimination order. Why? Because there are always leaf nodes available.
- For arbitrary graphs, must there be a perfect elimination order?

Non-trees: is there always a perfect elimination order?



- Left: Eliminating x_4 is bad, but other nodes are better.
- Left: No node results in zero fill in! ☹
- Right: Is there a perfect elimination order?
- For exact inference and some queries, inevitable that we work with a larger family since $\mathcal{F}((V,E),\mathcal{M}^{(\mathsf{f})}) \subset \mathcal{F}((V,E\cup F),\mathcal{M}^{(\mathsf{f})})$.
- Appears to be computational equivalence classes of families of models.

Non-tree graphs

Lemma 4.3.1

From a computational perspective, the reconstituted graph on which elimination has been run is the family on which we are running inference. If fill-in is caused by elimination, inference is solved on a family larger than that specified by the original graph, and we might as well have started with that family to begin with. If an elimination order produces no fill-in, we are solving the inference query optimally.

- Also, ordering σ matters. Using σ a second time results in a perfect elimination order.

Neighbors of v same in original and reconstituted graph

Lemma 4.3.2

When elimination is run for a second time on the reconstituted graph with the same order, the set of neighbors v at the time v is eliminated is the same in both the original and in the reconstituted graph.

Proof.

Any neighbor of v in the reconstituted graph must be either an original-graph edge, or it must be due to a fill-in edge between v and some other node that is not an original graph neighbor. All of the fill-in neighbors must be due to elimination of nodes before v since after v is eliminated no new neighbors can be added to v. But the point at which v is eliminated in the original graph and the point at which it v is eliminated in the reconstituted graph, the same previous set of nodes have been eliminated, so any neighbors of v in the reconstituted graph will have been already added to the original graph when v is eliminated in the original graph.

Complexity of elimination process

Lemma 4.3.3

Given an elimination order, the computational complexity of the elimination process is $O(r^{k+1})$ where k is the largest set of neighbors encountered during elimination. This is the size of the largest clique in the reconstituted graph.

Proof.

First, when we eliminate σ_i in G_{i-1}, eliminating variable v when it is in the context of its current neighbors will cost $O(r^\ell)$ where $\ell = |\delta_{G_{i-1}}(v) + 1|$ — thus, the overall cost will be $O(r^{k+1})$.
Next, we show that largest clique in the reconstituted graph is equal to the complexity. Consider the reconstituted graph, and assume its largest clique is of size $k+1$. When we re-run elimination on this graph, there will be no fill in. ...

Proof cont.

. . . continued.

However, the cost of the elimination step upon reaching the first vertex v of the clique of size $k+1$ will be $O(r^{k+1})$ since k of the variables of the clique will be neighbors of v, but no other nodes will be neighbors since it is a perfect elimination order in the reconstituted graph. This will be the same cost as what was incurred during the initial elimination procedure since v has the same set of neighbors. Therefore, the largest clique in the reconstituted graph is the complexity of doing elimination. □

- This means that any perfect elimination ordering on a perfect-elimination graph will have complexity exponential in the size of the largest clique in that graph.

Non-tree graphs

Summarizing what we've got so far:

- $G' = (V, E \cup F_\sigma)$ always has at least one perfect elimination order
- When we run elimination algorithm, we will always end up with such a graph - inevitable
- Perhaps we should deal only with such graphs?
- Is finding the order that minimizes fill-in optimal? (we shall see)
- We can characterize the complexity of a given elimination order.

Perfect elimination graphs

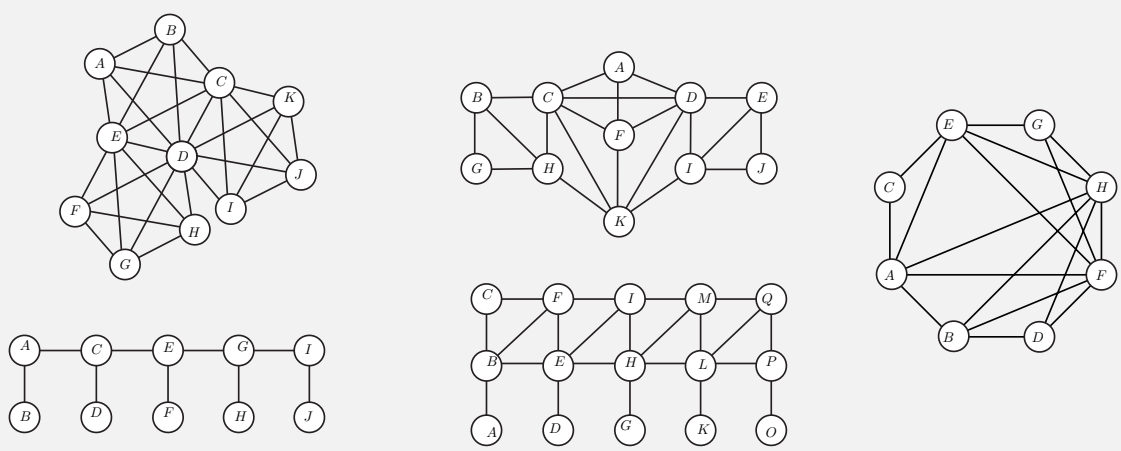
- Since such graphs are inevitable, lets define them and give them a name

Definition 4.4.1 (perfect elimination graph)

A graph $G = (V, E)$ is a *perfect elimination graph* if there exists an ordering σ of the nodes such that eliminating nodes in G based on σ produces no fill-in edges.

- any perfect elimination ordering on a perfect elimination graph will have complexity exponential in the size of the largest clique in that graph

Perfect elimination graphs?



Maxcliques of perfect elimination graphs

Lemma 4.4.2

When running the elimination algorithm, all maxcliques in the resulting reconstituted graph are encountered as elimination cliques during elimination.

Proof.

Each elimination step produces a clique, but not necessarily a maxclique. Set of maxcliques in the resulting reconstituted perfect elimination graph is a subset of the set of cliques encountered during elimination. This is because of the neighbor property proven above in Lemma 4.3.2 — if there was a maxclique in the reconstituted graph that was not one of the elimination cliques, that maxclique would be encountered on a run of elimination with the same order on the reconstituted graph, but for the first variable to encounter this maxclique, it would have the same set of neighbors in original graph, contradicting the fact that it was not one of the elimination cliques. □

Finding the maxcliques of G'

Lemma 4.4.3

Given a graph G, an order σ, and a reconstituted graph G', the elimination algorithm can produce the set of maxcliques in G'.

Proof.

Consider node v's elimination clique c_v (i.e., v along with its neighbors $\delta(v)$ at the time of elimination of v). Since c_v is complete, either c_v is a maxclique or a subset of some maxclique. c_v can not be a subset of any subsequently encountered maxcliques since all such future maxcliques would not involve v. Therefore c_v must be a maxclique or a subset of some previously encountered maxclique. If c_v is not a subset of some previously encountered maxclique, it must be a maxclique (we add c_v to a list of maxcliques). Since all maxcliques are encountered as elimination cliques, all maxcliques are discovered in this way. □

Finding maxcliques

Corollary 4.4.4

The first node eliminated in a graph, along with its neighbors, forms a maxclique.

- A node can be a member of more than 1 maxclique. Example, 4-cycle with diagonal edge. Is there a bound on the number of maxcliques a node might be a member of? Consider star tree graph.
- Inevitability: We have $p \in \mathcal{F}((V, E), \mathcal{M}^{(\mathsf{f})})$. We must work with $\mathcal{F}((V, E \cup F_\sigma), \mathcal{M}^{(\mathsf{f})})$.
- Q1: Can we identify the smallest such larger family (best elimination order σ) in which inference is solved?
- Q2: Does there exist a property (other than having a perfect elimination order) that characterizes this family of graphs?

Embedding

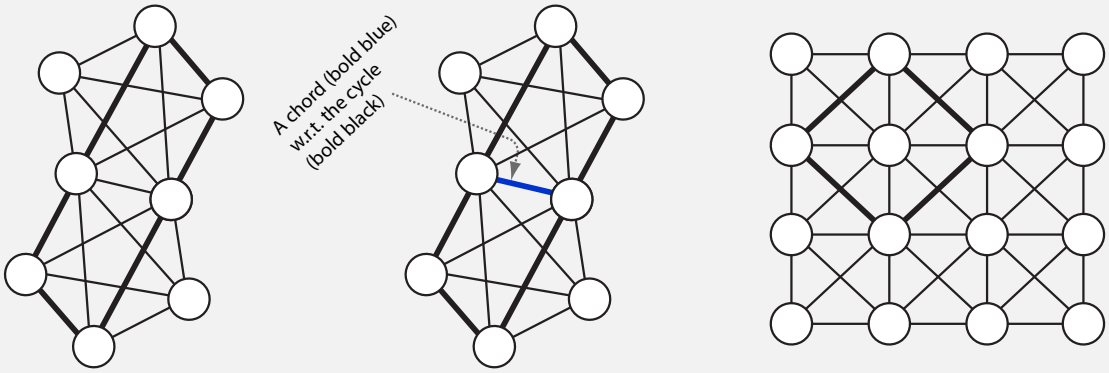
Definition 4.4.5 (embedding)

Any graph $G = (V, E)$ can be embedded into a graph $G' = (V, E')$ if G is a spanning subgraph of G', meaning that $E \subseteq E'$.

- Embedding never shrinks family of distributions
- Any G may be embedded into G_σ.
- We wish to embed G into the class of perfect elimination graphs (this is a subset of all undirected graphs).
- Does this restrict us in any way? (e.g., remove family members?)
- Does it change values of resulting queries we wish to compute?
- No, only potential issue is computation.
- Graphical model structure learning would be: start with $p \in \mathcal{F}(G, \mathcal{M}^{(\mathsf{f})})$, find some spanning subgraph $G' = (V, E')$ where $E' \subset E$, and solve inference there for a $p' \in \mathcal{F}(G', \mathcal{M}^{(\mathsf{f})})$ that is as close as possible to p. We defer this topic until later in the course.

Triangulated Graphs

- Triangulated graphs are also sometimes referred to either as *chordal*, *rigid-circuit*, *monotone transitive*, or (as we saw above) *perfect elimination* graphs.
- A *chord*, with respect to a cycle in a graph G, is an edge that directly connects two non-adjacent nodes in that cycle.



Triangulated Graphs

Definition 4.5.1 (Triangulated graph)

A graph G is triangulated (equivalently chordal) if all cycles have a chord.

- in triangulated graph: any cycles of length > 3 must have a chord.
- Cycles of length 3 have no non-adjacent vertices
- Triangulated graphs include
  1. a clique is a triangulated graph (all cycles have chord).
  2. a tree is a triangulated graph, since there are no cycles that could disobey the chordal requirement.
  3. a chain is a triangulated graph, since it is a tree.
  4. a set of disconnected vertices is triangulated (since there are no cycles).

Triangulated Graphs

Lemma 4.5.2 (Hereditary property of triangulated graphs)

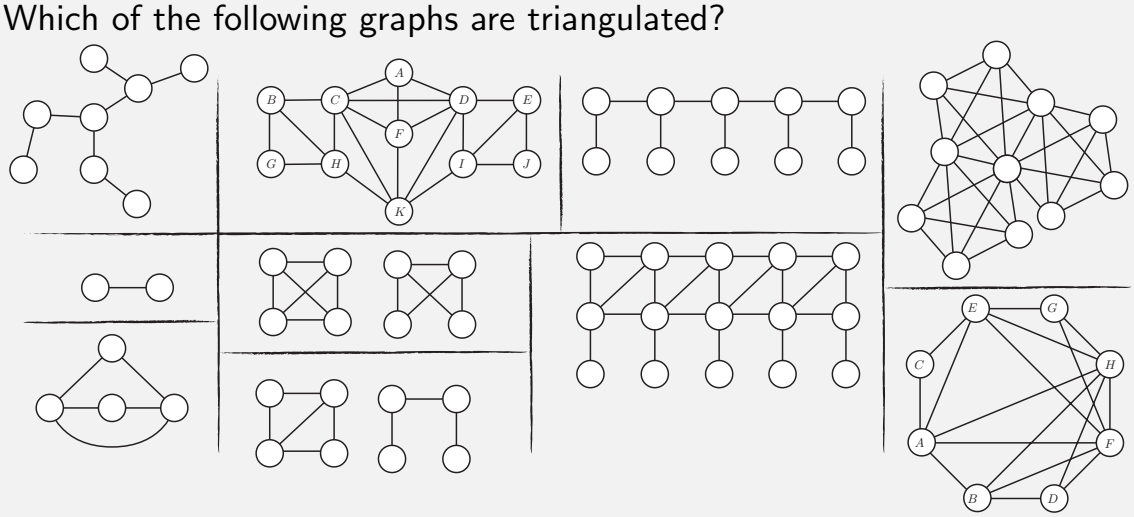
Any node-induced sub-graph of a triangulated graph is a triangulated graph.

Proof.

If a graph has no chordless cycles, then it has no chordless cycles involving any node v, and removing v only removes cycles involving v and so does not create any new chordless cycles. □

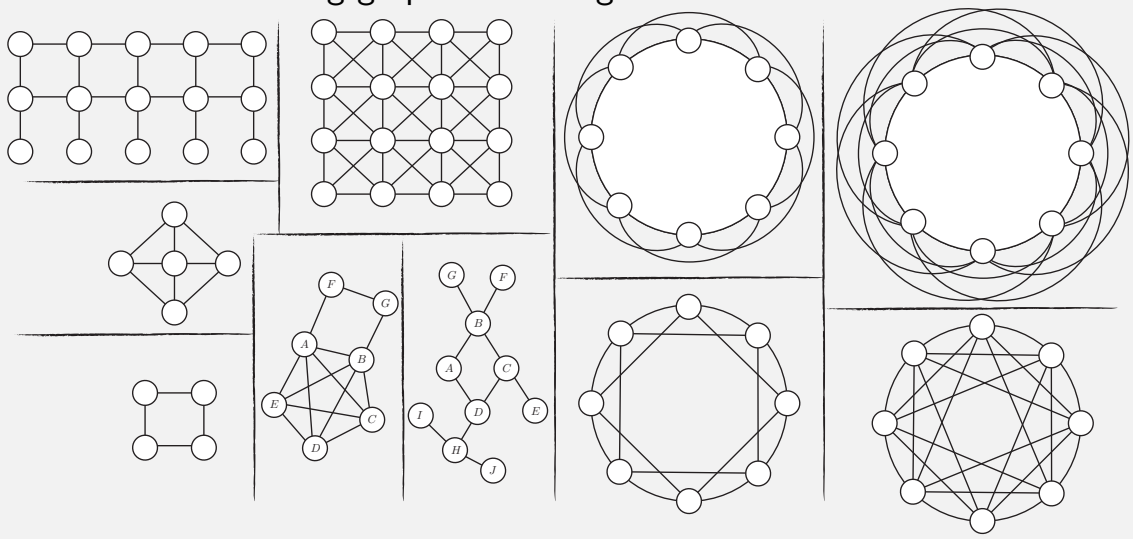
Triangulated Graphs

Which of the following graphs are triangulated?



Triangulated Graphs

Which of the following graphs are triangulated?



Triangulated Graphs

- Nodes that have no fill-in have a special name

Definition 4.5.3 (Simplicial)

Let $\delta(v) = \{u : (u, v) \in E(G)\}$ be the set of node neighbors of v in $G = (V, E)$. Then we say that v is simplicial if the vertex induced subgraph $G[\delta(v)]$ is a complete graph.

Triangulated Graphs

Theorem 4.5.4

Given graph G, elimination order σ, and perfect elimination graph $G' = G_\sigma$ obtained by elimination on G. We may reconstruct a perfect elimination order (w.r.t. G_σ) from G_σ by repeatedly choosing any simplicial node and eliminating it. Call this new order σ'. Now σ' might not be the same order as σ, but both are perfect elimination orders for G'.

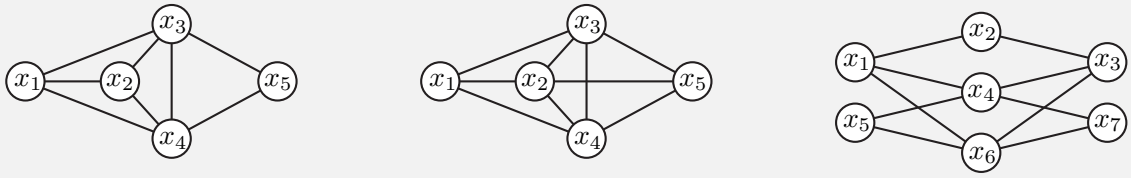
Proof.

If there is more than one possible order, we must reach a point at which there are two possible simplicial nodes $u, v \in G'$. Eliminating u does not render v non-simplicial since no edges are added and thus v has if anything only a reduced set of neighbors. Each time we eliminate a simplicial node, any other node that was simplicial in the original elimination order stays simplicial when it comes time to eliminate it. □

Graph separators

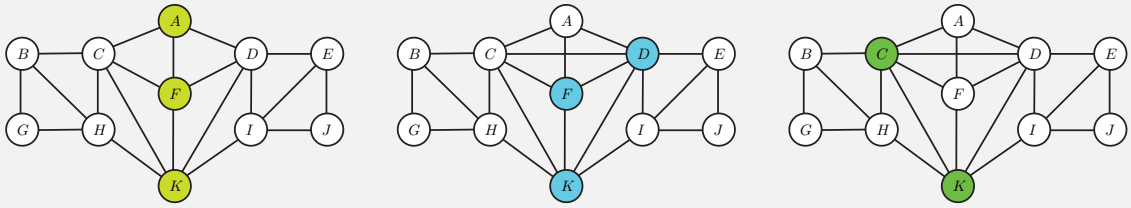
- Given $a, b \in V$, $a \neq b$, a set $S \subseteq V$ is an **(a, b)-separator** in G, if all paths from a to b must intersect some node in S.
- A **minimal (a, b)-separator** S is an (a, b)-separator such that any strict subset $S' \subset S$ is no longer an (a, b)-separator.
- A set S is a **separator** in $G = (V, E)$ if there exists $a, b \in V$ such that S is an (a, b)-separator.
- a set S is a **minimal separator** if there exists an $a, b \in V$ such that S is a minimal (a, b)-separator.

Graph separators - examples



- Left: both $\{x_3, x_4\}$ and $\{x_2, x_3, x_4\}$ a (x_1, x_5)-separator, only $\{x_3, x_4\}$ is a minimal (x_1, x_5)-separator.
- Middle: $\{x_3, x_4\}$ no longer a separator. $\{x_2, x_3, x_4\}$ now a minimal (x_1, x_5)-separator.
- Right: $\{x_2, x_4, x_6\}$ minimal (x_1, x_3)-separator, $\{x_4, x_6\}$ is a minimal (x_5, x_7)-separator.

Graph separators - examples



- Left: A, F, K is a minimal (B, E)-separator.
- Middle: D, F, K is a non-minimal (B, E)-separator
- Right: C, K is a minimal (B, E)-separator

Graph separators - examples

Lemma 4.5.5

Let S be a minimal (a, b)-separator in $G = (V, E)$ and let G_A, G_B be the connected components of G once S is removed containing a and b (i.e., $(G_A, G_B) \subseteq G[V \setminus S]$) where $a \in V(G_A)$ and $b \in V(G_B)$. Then each $s \in S$ is adjacent both to some node in G_A and some node in G_B.

Proof.

Suppose the contrary, that there exists an $s \in S$ not adjacent to any $v \in G_A$. In such case, $S \setminus \{s\}$ is still an (a, b)-separator since no path from G_A can get directly through s, contradicting the minimality of S. □

Triangulated graphs and minimal separators

Lemma 4.5.6

A graph $G = (V, E)$ is triangulated iff all minimal separators are complete.

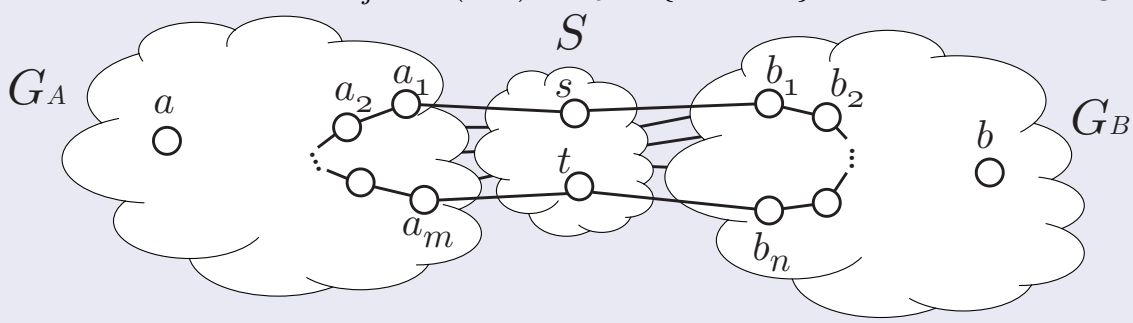
Proof.

First, suppose all minimal separators in $G = (V, E)$ are complete. Consider any cycle $u, v, w, x_1, x_2, \ldots, x_k, u$ starting and ending at node u, where $k \geq 1$. Then the pair v, x_i for some $i \in \{1, \ldots, k\}$ must be part of a minimal (u, w)-separator, which is complete, so v and that x_i are connected thereby creating a chord in the cycle. The cycle is arbitrary, so all cycles are chorded. ...

Triangulated graphs and separators

... proof continued.

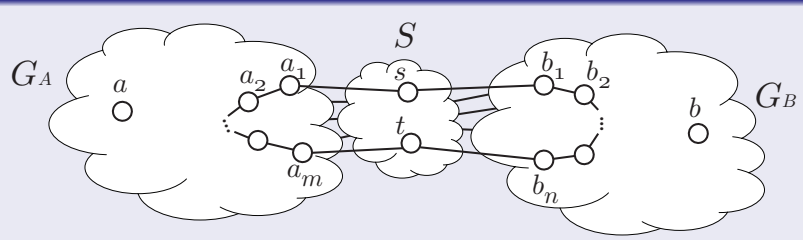
Next, suppose $G = (V, E)$ is triangulated, and let S be a minimal (a, b)-separator in G, and let G_A and G_B be the *connected* components of $G[V \setminus S]$ containing respectively a and b. Each $s \in S$ is connected to some $u \in V(G_A)$ and $v \in V(G_B)$. Therefore, since the components are connected, for each $s, t \in S$, there is a shortest path $s, a_1, a_2, \ldots, a_m, t$ with $a_i \in V(G_A)$ for $i \in \{1, \ldots, m\}$, and a shortest path $t, b_1, b_2, \ldots, b_n, s$ with $b_j \in V(G_B)$ for $j \in \{1, \ldots, n\}$, as in the following:



...

Triangulated graphs and separators

... proof continued.



Only successive a_i's in the path, and also s, a_1 and a_m, t, are adjacent as otherwise the path could be made shorter. The corresponding property is also true for the b_i's. Also, no a_i is adjacent to any b_i since S is a separator. A cycle is formed by $s, a_1, a_2, \ldots, a_m, t, b_1, a_2, \ldots, a_n, s$ which must have a chord, and the only candidate chord left is s, t. Since s, t are chosen arbitrarily from S, all pairs of nodes in the minimal separator are connected, and it is thus complete. □

Sources for Today's Lecture

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