EE512A – Advanced Inference in Graphical Models

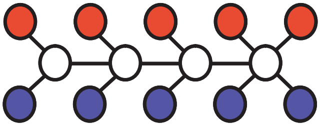
— Fall Quarter, Lecture 4 —

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, Inference on Trees
- L3 (10/6): Tree inference, more general queries, non-trees.
- L4 (10/8): Non-trees, perfect elimination, triangulated graphs, multiple queries
- L5 (10/13):
- 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.

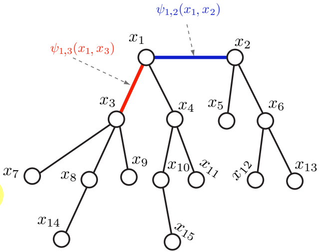
Generic form of message

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

Message is of form:

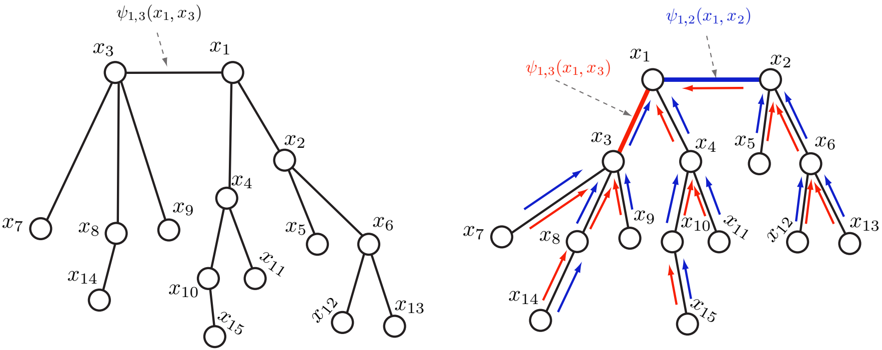
1. First, collect messages from all neighbors of i other than j,
2. next, incorporate these incoming messages by multiplying them in along with the factor $\psi_{i,j}(x_i, x_j)$,
3. the factor $\psi_{i,j}(x_i, x_j)$ relates x_i and x_j, and can be seen as a representation of a "communications channel" 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

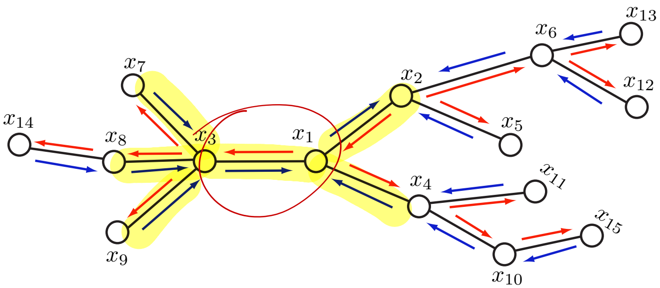


.

- 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

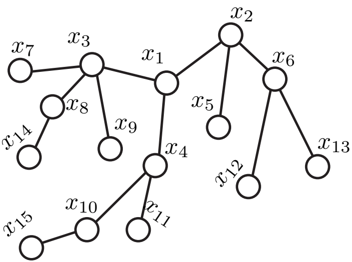
- As number of queries increases, so does efficiency (queries/message)
- Consider computing $p(x_i, x_j)$ for all $(i, j) \in E(G)$.
- Naive case, $N - 1$ edges $O(N^2 r^2)$.
- Smart case, only $O(N r^2)$ still.
- consider: root tree at all $(i, j) \in E(G)$ in turn
- mark edge with arrow only once (so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

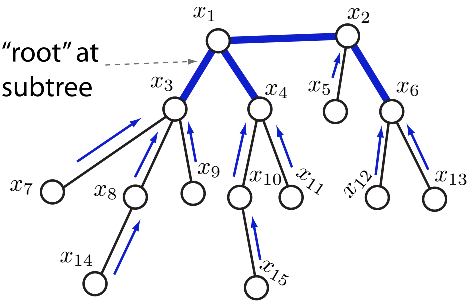


Collect/Distribute Evidence and MPP

- 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

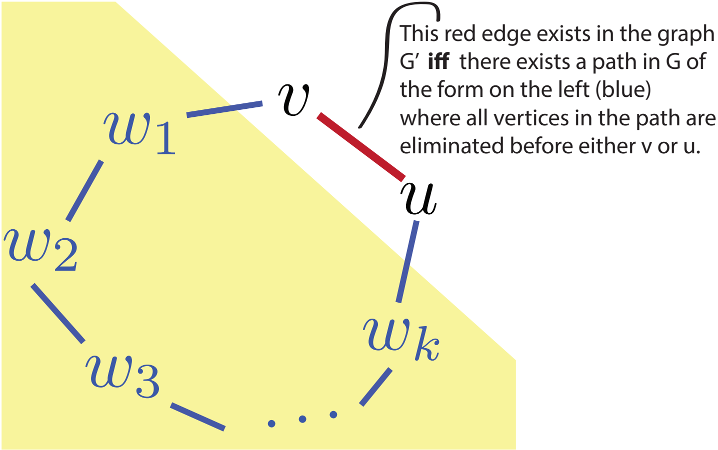
- We eliminate $x_{V\setminus S}$, which might introduce edges.
- Let $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_N)$ be an ordering of the nodes. Also $\sigma^{-1}(v)$ for $v \in V(G)$ gives number that node v is eliminated by order σ. We have following theorem

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 $(v = v_1, v_2, \dots, v_{k+1} = w)$ 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.

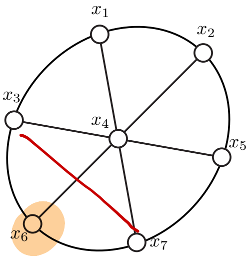
Perfect elimination orders

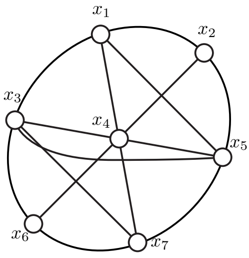
Definition 4.2.2 (perfect elimination order)

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.

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.

Non-tree graphs

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.

Non-tree graphs

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.

Non-tree graphs

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.

$k+1$

Proof.

σ_i'

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. $\square$

- 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

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

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?

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)

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

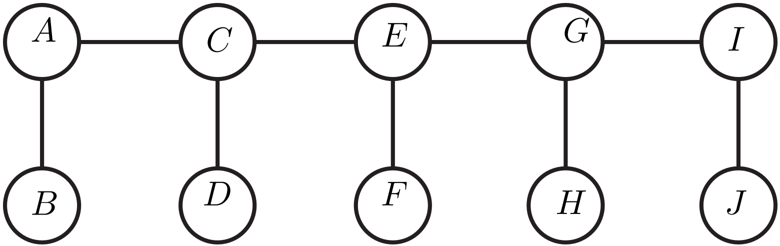
- Since such graphs are inevitable, lets define them and give them a name

Definition 4.4.1 (perfect elimination graph)

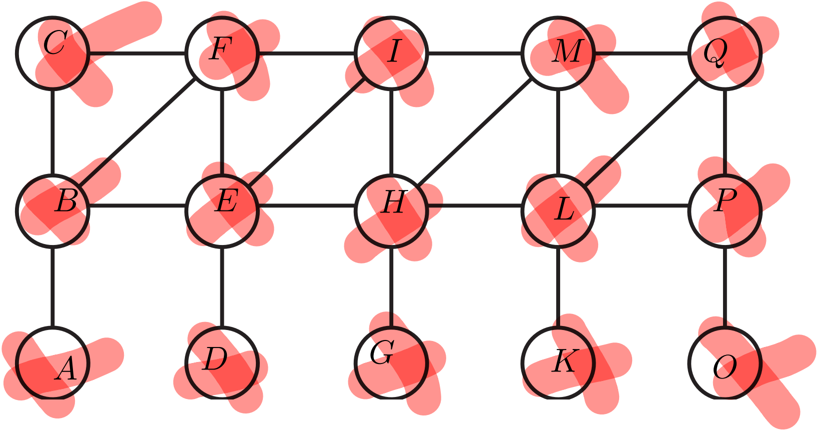
A graph $G = (V, E)$ is a *perfect elimination graph* if there exists an ordering σ of the nodes such that eliminating nodes in G based on σ produces no fill-in edges.

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

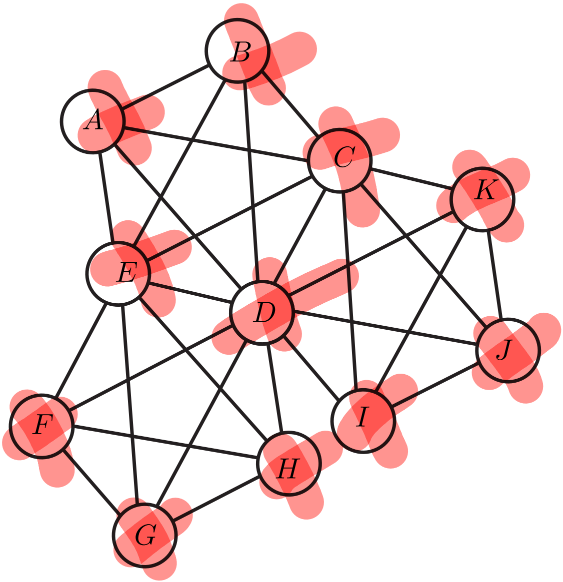
Perfect elimination graphs?



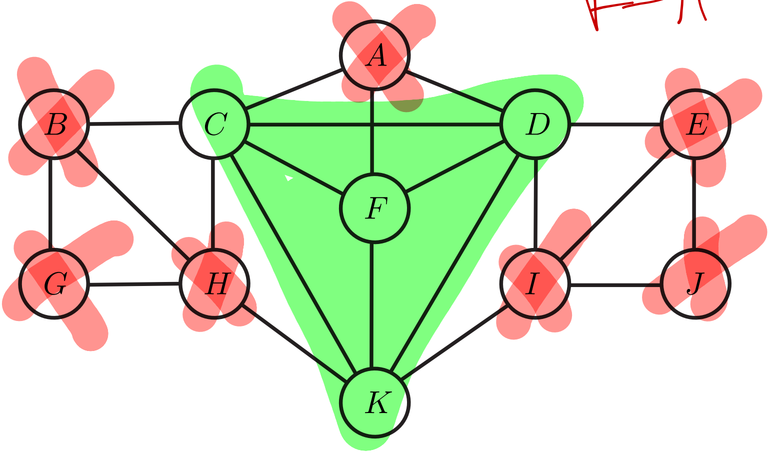
Perfect elimination graphs?



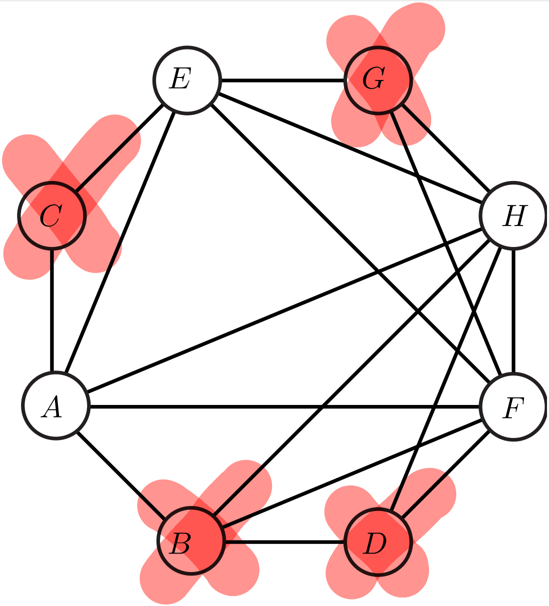
Perfect elimination graphs?



Perfect elimination graphs?



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

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.

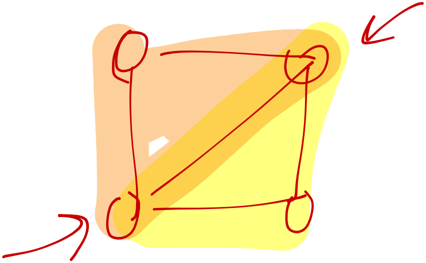
Consider node v's elimination clique c_v (i.e., v along with its neighbors $\delta(v)$ at the time of elimination of v). Since c_v is complete, either c_v is a maxclique or a subset of some maxclique. c_v can not be a subset of any subsequently encountered maxcliques since all such future maxcliques 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. □

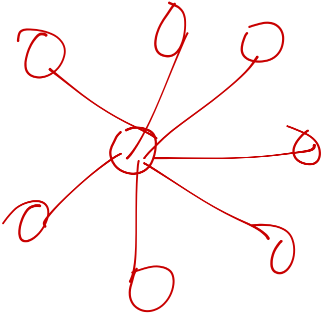
Finding maxcliques

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?





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

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.

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 $E \subseteq E'$.

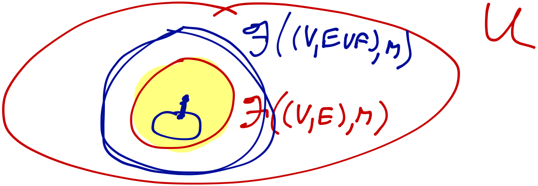
- Embedding never shrinks family of distributions
- Any G may be embedded into G_σ.
- We wish to embed G into the class of perfect elimination graphs (this is a subset of all undirected graphs).

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?)



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?

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.

Embedding

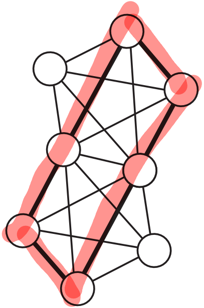
Definition 4.4.5 (embedding)

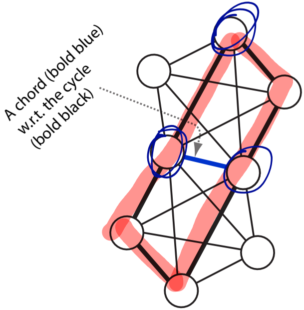
Any graph $G = (V, E)$ can be embedded into a graph $G' = (V, E')$ if G is a spanning subgraph of G', meaning that $E \subseteq E'$.

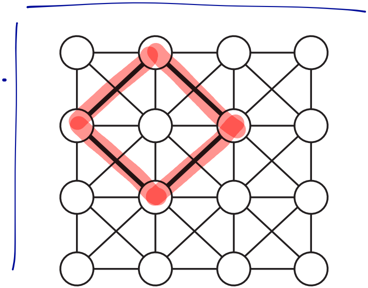
- Embedding never shrinks family of distributions
- Any G may be embedded into G_σ.
- We wish to embed G into the class of perfect elimination graphs (this is a subset of all undirected graphs).
- Does this restrict us in any way? (e.g., remove family members?)
- Does it change values of resulting queries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

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

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

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).

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.

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.

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)

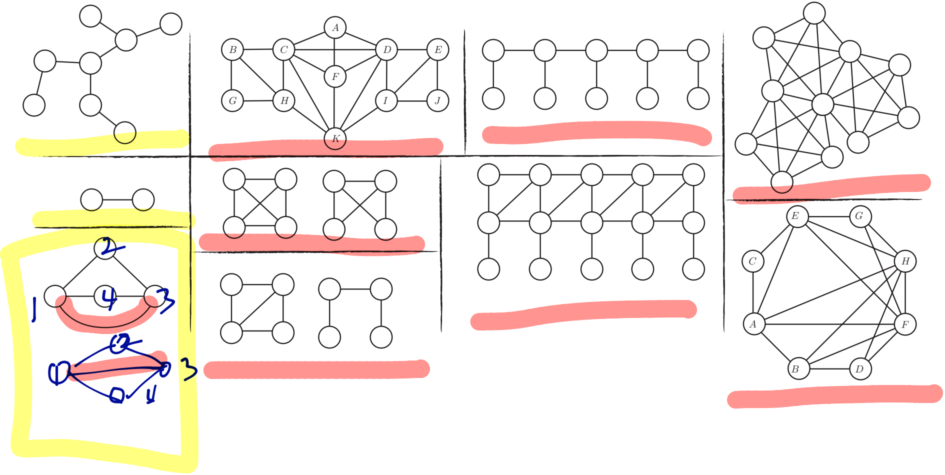
Any node-induced sub-graph of a triangulated graph is a triangulated graph.

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. □

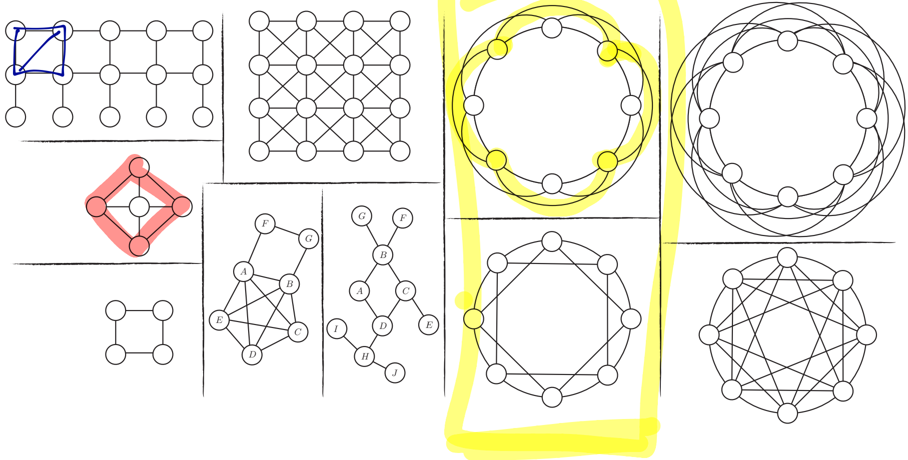
Triangulated Graphs

Which of the following graphs are triangulated?

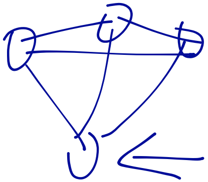


Triangulated Graphs

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

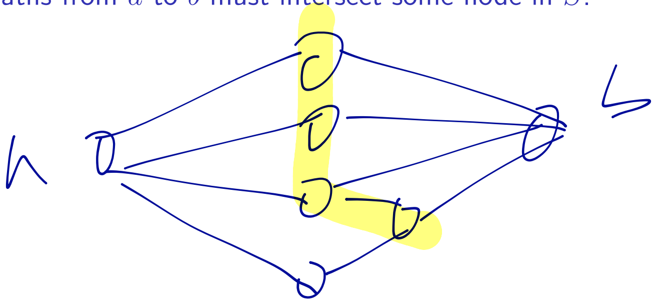
Given graph G, elimination order σ, and perfect elimination graph $G' = G_\sigma$ obtained by elimination on G. We may reconstruct a perfect elimination order (w.r.t. G_σ) from G_σ by repeatedly choosing any simplicial node and eliminating it. Call this new order σ'. Now σ' might not be the same order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

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.



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.

Graph separators

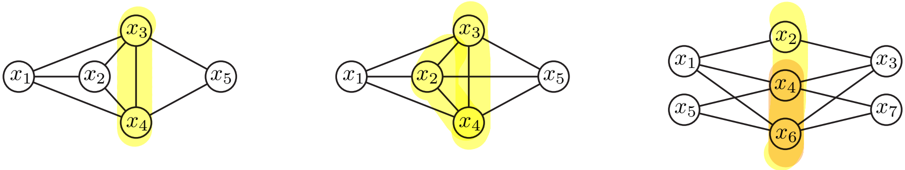
$a, b \notin S$

- 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.

Graph separators

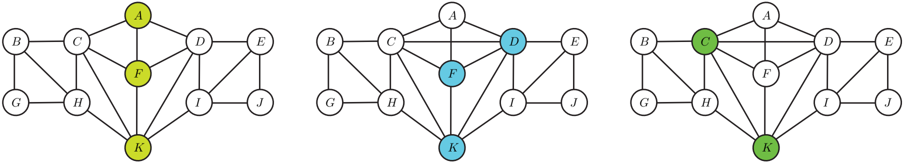
- Given $a, b \in V$, $a \neq b$, a set $S \subseteq V$ is an **(a, b)-separator** in G, if all paths from a to b must intersect some node in S.
- A **minimal (a, b)-separator** S is an (a, b)-separator such that any strict subset $S' \subset S$ is no longer an (a, b)-separator.
- A set S is a **separator**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

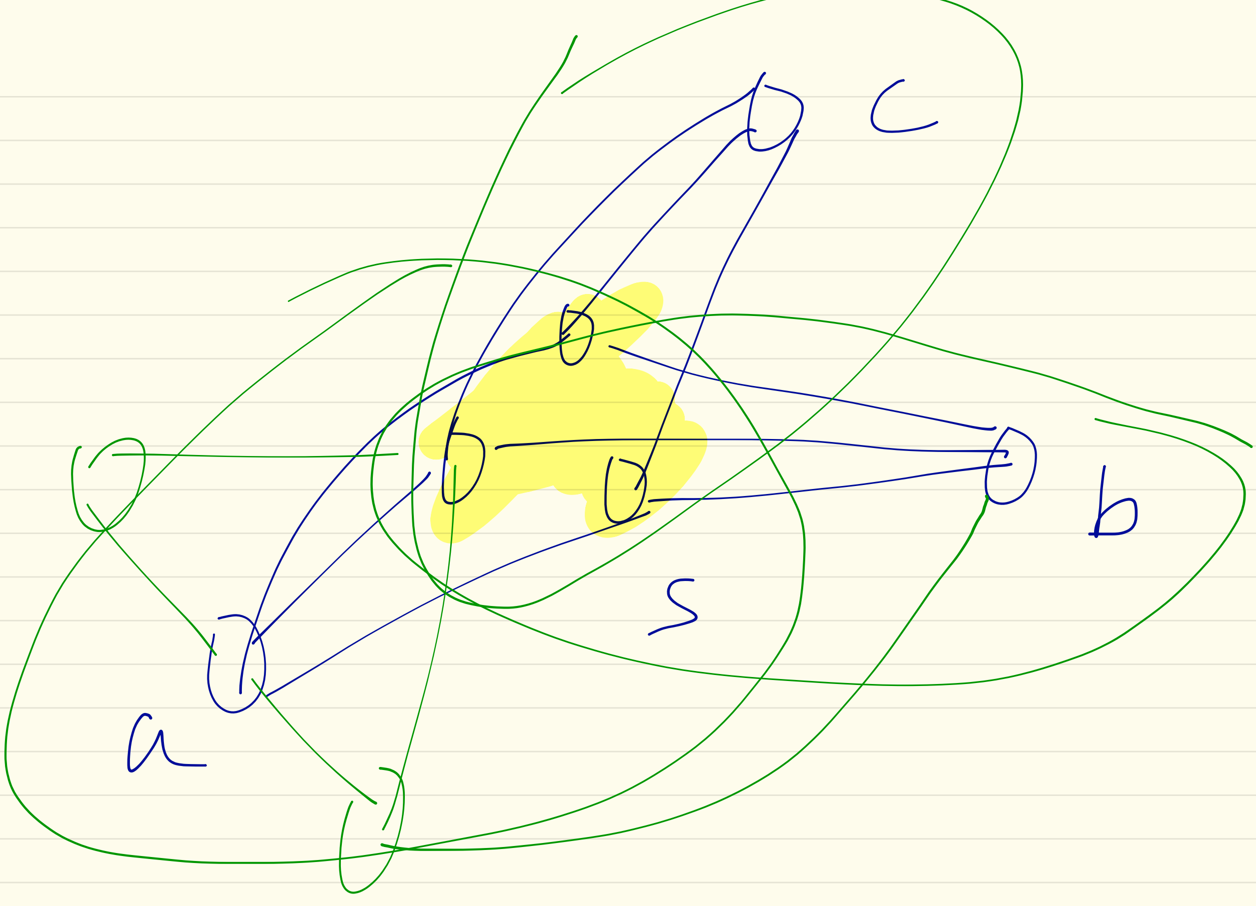


- Left: both $\{x_3, x_4\}$ and $\{x_2, x_3, x_4\}$ a (x_1, x_5)-separator, only $\{x_3, x_4\}$ is a minimal (x_1, x_5)-separator.
- Middle: $\{x_3, x_4\}$ no longer a separator. $\{x_2, x_3, x_4\}$ now a minimal (x_1, x_5)-separator.
- Right: $\{x_2, x_4, x_6\}$ minimal (x_1, x_3)-separator, $\{x_4,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

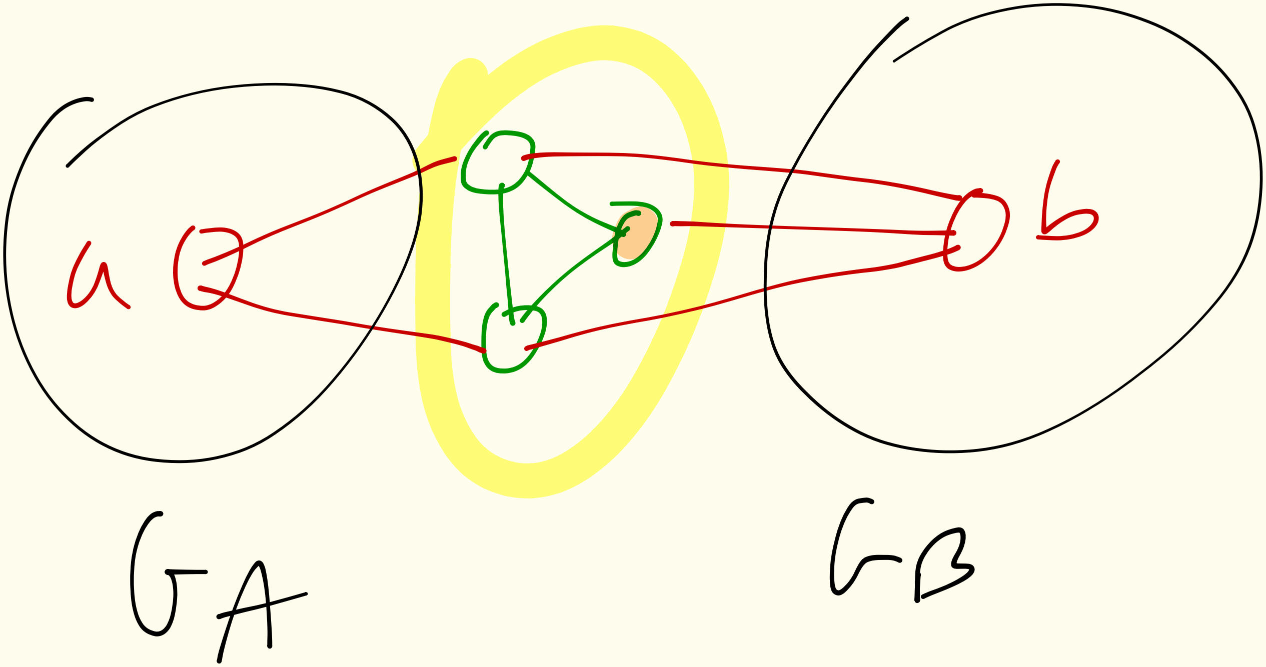
Graph separators - examples



- 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



c
b
a
S


a
b
GA
GB

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. □

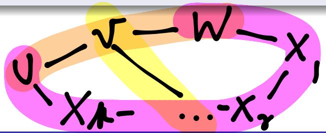
Triangulated graphs and separators

Lemma 4.5.6

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

Proof.

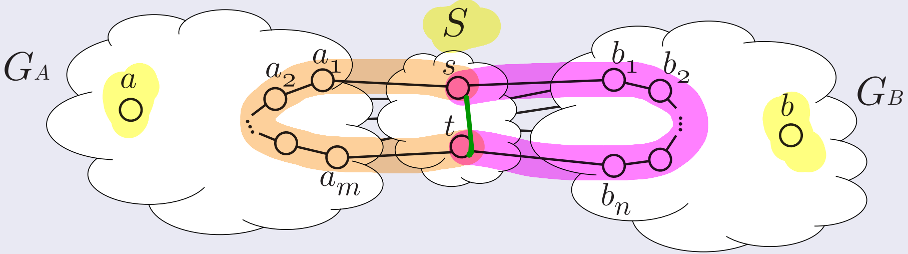
First, suppose all minimal separators in $G = (V, E)$ are complete. Consider any cycle $u, v, w, x_1, x_2, \ldots, x_k, u$ starting and ending at node u, where $k \geq 1$. Then the pair v, x_i for some $i \in \{1, \ldots, k\}$ must be part of a minimal (u, w)-separator, which is complete, so v and that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

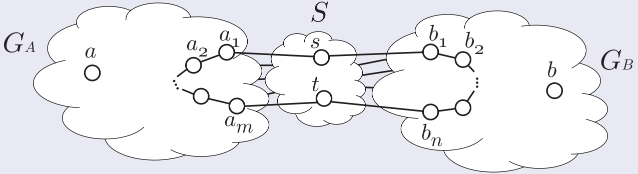
Next, suppose $G = (V, E)$ is triangulated, and let S be a minimal (a, b)-separator in G, and let G_A and G_B be the *connected* components of $G[V \setminus S]$ containing respectively a and b. Each $s \in S$ is connected to some $u \in V(G_A)$ and $v \in V(G_B)$. Therefore, since the components are 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. □

Triangulated graphs and separators

G triangulated means all (a, b)-separators are complete.

We also have the following important theorem.

Lemma 4.5.7

A triangulated graph on $n \geq 2$ nodes is either a clique, or there are two non-adjacent nodes that are simplicial.

Note that this appears to be very much like the property of a tree where a simplicial node takes the role of a leaf-node. Is this a coincidence?

Triangulated graphs have at least two simplicial nodes.

Proof.

Any clique is triangulated and all nodes are simplicial, so assume the graph is not a clique. Induction on $n = |V(G)|$: any graph with $1 < n \leq 3$ is triangulated and has two simplicial nodes. Assume true for $n - 1$ nodes, and show for n nodes. Let a and b be two non-adjacent vertices, let S be a minimal (a, b)-separator which must be complete. Let G_A and G_B be the connected components of $G[V \setminus S]$ containing respectively a and b. Let $A = V(G_A)$ and $B = V(G_B)$. By induction, $G[A \cup S]$ and $G[B \cup S]$ are either cliques, or contain two non-adjacent simplicial vertices. First case, all nodes are simplicial, second case both simplicial non-adjacent vertices cannot be in S since S is complete. In all cases, we may choose two non-adjacent simplicial vertices, one each in A and B, and these vertices are adjacent to no nodes other than $A \cup S$ and $B \cup S$ respectively. These nodes remain simplicial and non-adjacent in G. $\square$

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.
- Elimination cliques are supserset of set of maxcliques.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.
- Elimination cliques are supserset of set of maxcliques.
- We may embed any $G = (V, E)$ into any $G = (V, E \cup F)$.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.
- Elimination cliques are supserset of set of maxcliques.
- We may embed any $G = (V, E)$ into any $G = (V, E \cup F)$.
- Given perfect elimination graph, easy to find perfect elimination order.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.
- Elimination cliques are supserset of set of maxcliques.
- We may embed any $G = (V, E)$ into any $G = (V, E \cup F)$.
- Given perfect elimination graph, easy to find perfect elimination order.
- Triangulated graphs (chordal), all cycles are chorded.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.
- Elimination cliques are supserset of set of maxcliques.
- We may embed any $G = (V, E)$ into any $G = (V, E \cup F)$.
- Given perfect elimination graph, easy to find perfect elimination order.
- Triangulated graphs (chordal), all cycles are chorded.
- Various definitions of separators.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.
- Elimination cliques are supserset of set of maxcliques.
- We may embed any $G = (V, E)$ into any $G = (V, E \cup F)$.
- Given perfect elimination graph, easy to find perfect elimination order.
- Triangulated graphs (chordal), all cycles are chorded.
- Various definitions of separators.
- Triangulated iff all min separators are complete.

Recap

- Non-tree graphs: effectively doing inference on perfect elimination graph.
- After elimination, we've got a perfect (fill-in free) elimination graph.
- We encounter the maxcliques when we run elimination.
- Elimination cliques are supserset of set of maxcliques.
- We may embed any $G = (V, E)$ into any $G = (V, E \cup F)$.
- Given perfect elimination graph, easy to find perfect elimination order.
- Triangulated graphs (chordal), all cycles are chorded.
- Various definitions of separators.
- Triangulated iff all min separators are complete.
- Any triangulated graph on ≥ 2 nodes has two simplicial nodes.

Triangulated/Elimination

- In a triangulated graphs, all nodes simplicial?

Triangulated/Elimination

- In a triangulated graphs, all nodes simplicial?
- If G is triangulated and v simplicial, if we eliminate v, is $G[V \setminus v]$ still triangulated?

Triangulated/Elimination

- In a triangulated graphs, all nodes simplicial?
- If G is triangulated and v simplicial, if we eliminate v, is $G[V \setminus v]$ still triangulated?
- Therefore:

Corollary 4.5.8

For any triangulated graph, there exists an elimination order that does not produce any fill in.

So if we know the graph is triangulated, we can easily find a perfect elimination order. Why?

Triangulated/Elimination

- In a triangulated graphs, all nodes simplicial?
- If G is triangulated and v simplicial, if we eliminate v, is $G[V \setminus v]$ still triangulated?
- Therefore:

Corollary 4.5.8

For any triangulated graph, there exists an elimination order that does not produce any fill in.

So if we know the graph is triangulated, we can easily find a perfect elimination order. Why? We can strengthen the above in fact:

Triangulated vs. Perfect elimination graphs

Lemma 4.5.9

If G is a graph and there exists a perfect elimination order, then G is triangulated.

Triangulated vs. Perfect elimination graphs

Lemma 4.5.9

If G is a graph and there exists a perfect elimination order, then G is triangulated.

Proof.

By induction. It is obviously true for 1 and 2 nodes. Assume true for n nodes, and we are given an $n+1$ node graph. Since there exists an elimination order without fill-in, there exists a simplicial node, where chordless cycles can not exist through that node since all of its neighbors are connected. Once we eliminate that node, no fill-in is created, and induction step applies. □

Triangulated vs. Perfect elimination graphs

Lemma 4.5.9

If G is a graph and there exists a perfect elimination order, then G is triangulated.

Proof.

By induction. It is obviously true for 1 and 2 nodes. Assume true for n nodes, and we are given an $n+1$ node graph. Since there exists an elimination order without fill-in, there exists a simplicial node, where chordless cycles can not exist through that node since all of its neighbors are connected. Once we eliminate that node, no fill-in is created, and induction step applies. □

We summarize the bijection as follows:

Triangulated vs. Perfect elimination graphs

Lemma 4.5.9

If G is a graph and there exists a perfect elimination order, then G is triangulated.

Proof.

By induction. It is obviously true for 1 and 2 nodes. Assume true for n nodes, and we are given an $n+1$ node graph. Since there exists an elimination order without fill-in, there exists a simplicial node, where chordless cycles can not exist through that node since all of its neighbors are connected. Once we eliminate that node, no fill-in is created, and induction step applies. □

We summarize the bijection as follows:

Theorem 4.5.10

A graph G is triangulated iff there exists a perfect elimination order over the nodes in G.

Triangulated vs. Perfect elimination graphs

Corollary 4.5.11

Take any graph G and an elimination order σ, then the reconstituted graph $G' = (V, E \cup F_\sigma)$ is triangulated.

Triangulated vs. Perfect elimination graphs

Generating triangulated graphs

- Therefore, we can generate a reconstituted elimination graph (or any triangulated graph) using a reverse elimination order.

Algorithm 9: Regenerate triangulated graph.

Input: A triangulated graph $G = (V, E)$ and a perfect elimination order σ
Result: A new graph G' identical to G.

```
Recall that δ_{G_{i-1}}(σ_i) are neighbors of σ_i in G at the point σ_i is eliminated. ;
Start out with V(G') empty ;
Add σ_N to V(G') ;
for i = N − 1 . . . 1 do
    Add σ_i to V(G') ;
    Add δ_{G_{i-1}}(σ_i) to E(G') ;        /* at this δ_{G_{i-1}}(σ_i) is complete */
```

Triangulated vs. Perfect elimination graphs

- Trees can be generated this way (recall one of the definitions)
- Does elimination span the space of all possible triangulations of a graph? (i.e., can any triangulation of G be obtained by some elimination order?)

Triangulated vs. Perfect elimination graphs

- Trees can be generated this way (recall one of the definitions)
- Does elimination span the space of all possible triangulations of a graph? (i.e., can any triangulation of G be obtained by some elimination order?)

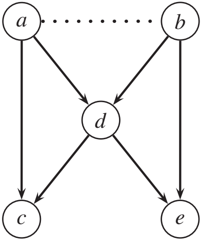
Theorem 4.5.12

Let $G = (V, E)$ be a graph and let $G' = (V, E \cup F)$ be a triangulation of G with F the required edge fill-in. If the triangulated graph is minimal *in the sense that for any $F' \subset F$, the graph $G'' = (V, E \cup F')$ is no longer triangulated, then F can be obtained by the result of an elimination order. That is, the elimination algorithm and the various variable orderings may produce all minimal triangulations of a graph G.*

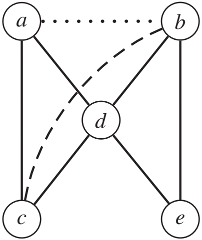
- Minimal triangulations are state-space optimal for positive distributions only!

Triangulated vs. Perfect elimination graphs

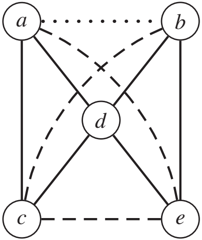
Minimal triangulations are state-space optimal for positive distributions only. Let d be a deterministic function of a and b. All variables have r values but d has $r^2 - 1$ values.



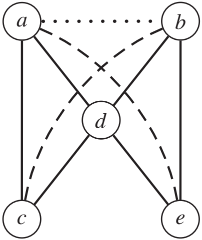
Moralized already chordal, perfect elim. order (c, a, b, d). One clique at $O(r^2)$, two at $O(r^4)$.



Elimination order (a, c, b, d), cost is still $O(r^4)$



Start by eliminating d, cost is still $O(r^4)$



Triangulation unobtainable with elimination, cost $O(r^3)$.

re-cap

- We wish to run elimination

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination
- Elimination adds edges, we can embed original graph into resulting triangulated graph (triangulated graph covers original graph)

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination
- Elimination adds edges, we can embed original graph into resulting triangulated graph (triangulated graph covers original graph)
- can't avoid a triangulated graph — always dealing with triangulated graphs implicitly or explicitly

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination
- Elimination adds edges, we can embed original graph into resulting triangulated graph (triangulated graph covers original graph)
- can't avoid a triangulated graph — always dealing with triangulated graphs implicitly or explicitly
- want to find covering minimally triangulated graph with smallest largest maxclique

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination
- Elimination adds edges, we can embed original graph into resulting triangulated graph (triangulated graph covers original graph)
- can't avoid a triangulated graph — always dealing with triangulated graphs implicitly or explicitly
- want to find covering minimally triangulated graph with smallest largest maxclique
- i.e., find optimal elimination order

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination
- Elimination adds edges, we can embed original graph into resulting triangulated graph (triangulated graph covers original graph)
- can't avoid a triangulated graph — always dealing with triangulated graphs implicitly or explicitly
- want to find covering minimally triangulated graph with smallest largest maxclique
- i.e., find optimal elimination order
- there are $n!$ elimination orders

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination
- Elimination adds edges, we can embed original graph into resulting triangulated graph (triangulated graph covers original graph)
- can't avoid a triangulated graph — always dealing with triangulated graphs implicitly or explicitly
- want to find covering minimally triangulated graph with smallest largest maxclique
- i.e., find optimal elimination order
- there are $n!$ elimination orders
- is this easy or hard?

re-cap

- We wish to run elimination
- Doing so produces a triangulated graph
- Complexity is its largest clique in result
- We encounter the cliques (and the largest) during elimination so we get the complexity while we are doing elimination
- Elimination adds edges, we can embed original graph into resulting triangulated graph (triangulated graph covers original graph)
- can't avoid a triangulated graph — always dealing with triangulated graphs implicitly or explicitly
- want to find covering minimally triangulated graph with smallest largest maxclique
- i.e., find optimal elimination order
- there are $n!$ elimination orders
- is this easy or hard? We shall see ...

k-trees

Generalizations of a tree as defined as follows:

Definition 4.5.13 (k-tree)

A complete graph with $k+1$ nodes is a k-tree. To construct a k tree with $n+1$ nodes starting from a k-tree with n nodes, choose some size k complete sub-graph of the n-node k-tree and connect the $n+1$'st node to all nodes in the k-node complete sub-graph.

k-trees

Generalizations of a tree as defined as follows:

Definition 4.5.13 (k-tree)

A complete graph with $k+1$ nodes is a k-tree. To construct a k tree with $n+1$ nodes starting from a k-tree with n nodes, choose some size k complete sub-graph of the n-node k-tree and connect the $n+1$'st node to all nodes in the k-node complete sub-graph.

- Any complete n-graph is an $n-1$-tree

k-trees

Generalizations of a tree as defined as follows:

Definition 4.5.13 (k-tree)

A complete graph with $k+1$ nodes is a k-tree. To construct a k tree with $n+1$ nodes starting from a k-tree with n nodes, choose some size k complete sub-graph of the n-node k-tree and connect the $n+1$'st node to all nodes in the k-node complete sub-graph.

- Any complete n-graph is an $n-1$-tree
- a regular tree is a 1-tree.

k-trees

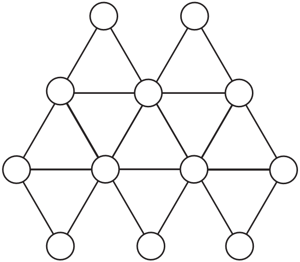
Generalizations of a tree as defined as follows:

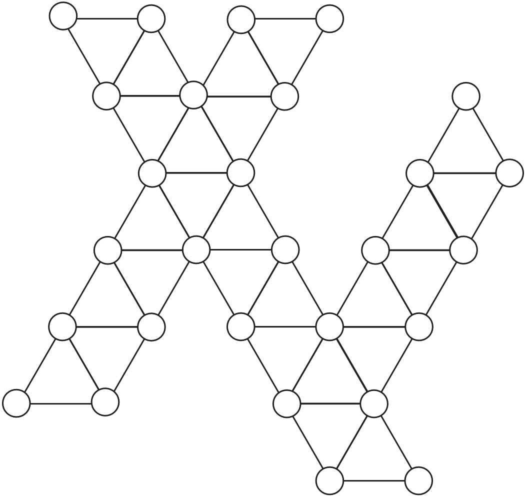
Definition 4.5.13 (k-tree)

A complete graph with $k+1$ nodes is a k-tree. To construct a k tree with $n+1$ nodes starting from a k-tree with n nodes, choose some size k complete sub-graph of the n-node k-tree and connect the $n+1$'st node to all nodes in the k-node complete sub-graph.

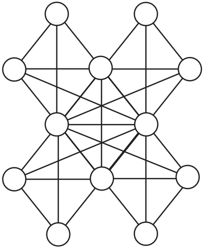
- Any complete n-graph is an $n-1$-tree
- a regular tree is a 1-tree.
- all k-trees are triangulated

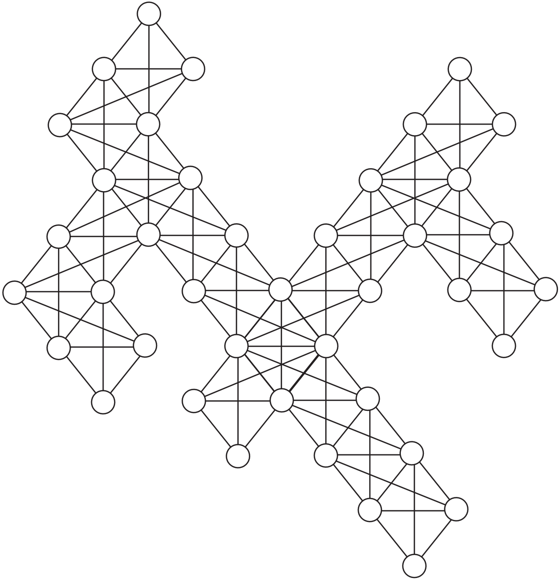
Example of 2-trees





Example of 3-trees





k-trees

- In a tree, all minimal separators are size 1

k-trees

- In a tree, all minimal separators are size 1
- In a k-tree, all minimal separators are size k (and thus a k-clique).

k-trees

- In a tree, all minimal separators are size 1
- In a k-tree, all minimal separators are size k (and thus a k-clique).
- In a k-tree, all maxcliques are size $k+1$, so maximum clique size is $k+1$.

k-trees

- In a tree, all minimal separators are size 1
- In a k-tree, all minimal separators are size k (and thus a k-clique).
- In a k-tree, all maxcliques are size $k+1$, so maximum clique size is $k+1$.
- In a k-tree, complexity of inference will be $O(r^{k+1})$.

k-trees

- In a tree, all minimal separators are size 1
- In a k-tree, all minimal separators are size k (and thus a k-clique).
- In a k-tree, all maxcliques are size $k+1$, so maximum clique size is $k+1$.
- In a k-tree, complexity of inference will be $O(r^{k+1})$.
- even stronger:

k-trees

- In a tree, all minimal separators are size 1
- In a k-tree, all minimal separators are size k (and thus a k-clique).
- In a k-tree, all maxcliques are size $k+1$, so maximum clique size is $k+1$.
- In a k-tree, complexity of inference will be $O(r^{k+1})$.
- even stronger:

Lemma 4.5.14

A graph $G = (V, E)$ is a k-tree iff

- *G is connected*
- *G's maximum clique is of size $k+1$*
- *Every minimal separator of G is a k-clique.*

k-trees

Definition 4.5.15 (partial k-tree)

Any spanning sub-graph of a k-tree is a partial k-tree.

- Any partial k-tree is embeddable into a k-tree.

k-trees

Definition 4.5.15 (partial k-tree)

Any spanning sub-graph of a k-tree is a partial k-tree.

- Any partial k-tree is embeddable into a k-tree.
- Inference in a partial k-tree is at most $O(r^{k+1})$.

k-trees

Definition 4.5.15 (partial k-tree)

Any spanning sub-graph of a k-tree is a partial k-tree.

- Any partial k-tree is embeddable into a k-tree.
- Inference in a partial k-tree is at most $O(r^{k+1})$.
- k-trees are triangulated, but arbitrary triangulated graph not necessarily a k-tree

k-trees

Definition 4.5.15 (partial k-tree)

Any spanning sub-graph of a k-tree is a partial k-tree.

- Any partial k-tree is embeddable into a k-tree.
- Inference in a partial k-tree is at most $O(r^{k+1})$.
- k-trees are triangulated, but arbitrary triangulated graph not necessarily a k-tree
- any triangulated graph can be embedded into a k tree for large enough k — silly example, set $k = (n - 1)$.

k-trees

Definition 4.5.15 (partial k-tree)

Any spanning sub-graph of a k-tree is a partial k-tree.

- Any partial k-tree is embeddable into a k-tree.
- Inference in a partial k-tree is at most $O(r^{k+1})$.
- k-trees are triangulated, but arbitrary triangulated graph not necessarily a k-tree
- any triangulated graph can be embedded into a k tree for large enough k — silly example, set $k = (n - 1)$.
- But is it possible for smaller k?

k-trees and embeddings

Lemma 4.5.16

If G is a triangulated graph with at least $k+1$ vertices and has a maximum clique of size at most $k+1$, then G can be embedded into a k-tree.

Proof.

Let $\sigma = (\sigma_1, \dots, \sigma_n)$ be perfect elim. order for G. We embed G into k-tree by adding edges to G so that same ordering is perfect in the k-tree. Induction.

Base case, any set of $k+1$ vertices can be embedded into a k tree by making those vertices a clique. Thus, add edges to last $k+1$ eliminated vertices, i.e., make $\{\sigma_{n-k}, \sigma_{n-k+1}, \dots, \sigma_n\}$ a $k+1$-clique. ...

cont.

... proof continued.

Induction: assume the subgraph with vertices $\{\sigma_{i+1}, \ldots, \sigma_n\}$ has been embedded into a k-tree T_{i+1}. Since the maximum clique size of G is $k+1$, in G vertex σ_i is adjacent to a clique c with no more than k vertices in $\{\sigma_{i+1}, \ldots, \sigma_n\}$. In the k-tree T_{i+1}, c is contained in a k-clique c'. When we make σ_i adjacent to all of the vertices of c', we obtain a k-tree T_i since σ_i is still simplicial in T_i. Repeating to σ_1 and result is supergraph of G with same order being perfect. □

k-trees and embeddings

- Therefore, reconstituted elimination graph can be embedded into a k-tree for large enough k.

k-trees and embeddings

- Therefore, reconstituted elimination graph can be embedded into a k-tree for large enough k.
- Note: in k-tree all cliques are size $k + 1$ but in our reconstituted perfect elimination graph, might only have one clique that is of size $k + 1$ (but even given this, we will still show a negative result on the next slide ☹)

k-trees and embeddings

- Therefore, reconstituted elimination graph can be embedded into a k-tree for large enough k.
- Note: in k-tree all cliques are size $k+1$ but in our reconstituted perfect elimination graph, might only have one clique that is of size $k+1$ (but even given this, we will still show a negative result on the next slide ☹)
- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree.

k-trees and embeddings

- Therefore, reconstituted elimination graph can be embedded into a k-tree for large enough k.
- Note: in k-tree all cliques are size $k+1$ but in our reconstituted perfect elimination graph, might only have one clique that is of size $k+1$ (but even given this, we will still show a negative result on the next slide ☹)
- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree.
- i.e.,find best "Chordal cover"

k-trees and embeddings

- Therefore, reconstituted elimination graph can be embedded into a k-tree for large enough k.
- Note: in k-tree all cliques are size $k+1$ but in our reconstituted perfect elimination graph, might only have one clique that is of size $k+1$ (but even given this, we will still show a negative result on the next slide ☹)
- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree.
- i.e.,find best "Chordal cover"

k-trees and embeddings

- Therefore, reconstituted elimination graph can be embedded into a k-tree for large enough k.
- Note: in k-tree all cliques are size $k+1$ but in our reconstituted perfect elimination graph, might only have one clique that is of size $k+1$ (but even given this, we will still show a negative result on the next slide ☹)
- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree.
- i.e.,find best "Chordal cover"
- Unfortunately:

k-trees and embeddings

- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree, i.e.,find best "Chordal cover"

k-trees and embeddings

- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree, i.e.,find best "Chordal cover"

Theorem 4.5.17

For an arbitrary graph $G = (V, E)$, finding the smallest k such that G can be embedded into a k-tree is an NP-complete optimization problem (i.e., the decision version of the problem, asking if G can be embedded into a k-tree of size k, is NP-complete).

k-trees and embeddings

- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree, i.e.,find best "Chordal cover"

Theorem 4.5.17

For an arbitrary graph $G = (V, E)$, finding the smallest k such that G can be embedded into a k-tree is an NP-complete optimization problem (i.e., the decision version of the problem, asking if G can be embedded into a k-tree of size k, is NP-complete).

- consider again elimination as summing out variables - not possible to guarantee optimal summation in poly-time order unless P=NP.

k-trees and embeddings

- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree, i.e.,find best "Chordal cover"

Theorem 4.5.17

For an arbitrary graph $G = (V, E)$, finding the smallest k such that G can be embedded into a k-tree is an NP-complete optimization problem (i.e., the decision version of the problem, asking if G can be embedded into a k-tree of size k, is NP-complete).

- consider again elimination as summing out variables - not possible to guarantee optimal summation in poly-time order unless P=NP.
- We resort to heuristics (min fill, min size, random chose from top ℓ with random restarts, etc. work well).

Heuristics for elimination

Since we can't expect to find a perfect elimination order, we have heuristics:

1. **min fill-in heuristic:**. Eliminate next the node n that would result in the smallest number of fill-in edges at that step. Break ties arbitrarily.

Heuristics for elimination

Since we can't expect to find a perfect elimination order, we have heuristics:

1. **min fill-in heuristic:**. Eliminate next the node n that would result in the smallest number of fill-in edges at that step. Break ties arbitrarily.
2. **min size heuristic:** Eliminate next the node that would result in the smallest clique when eliminated (i.e., choose the node as one with the smallest edge degree). Break ties arbitrarily.

Heuristics for elimination

Since we can't expect to find a perfect elimination order, we have heuristics:

1. **min fill-in heuristic:**. Eliminate next the node n that would result in the smallest number of fill-in edges at that step. Break ties arbitrarily.
2. **min size heuristic:** Eliminate next the node that would result in the smallest clique when eliminated (i.e., choose the node as one with the smallest edge degree). Break ties arbitrarily.
3. **min weight heuristic:** If the nodes have non-uniform domain sizes, then we choose next the node that would result in the clique with the smallest state space, which is defined as the product of the domain sizes. Break ties arbitrarily.

Better Heuristics for elimination

Variants and improvements to the above heuristics.

1. **tie-breaking:** When one heuristic has tie, choose one of the other heuristics to break tie.

Better Heuristics for elimination

Variants and improvements to the above heuristics.

1. **tie-breaking:** When one heuristic has tie, choose one of the other heuristics to break tie.
2. **non-greedy:** Rather than greedily choosing best vertex, take the m-best vertices (e.g., the $m < n$ nodes that would result in, say, the smallest fill-in) and eliminate one of them.

Better Heuristics for elimination

Variants and improvements to the above heuristics.

1. **tie-breaking:** When one heuristic has tie, choose one of the other heuristics to break tie.
2. **non-greedy:** Rather than greedily choosing best vertex, take the m-best vertices (e.g., the $m < n$ nodes that would result in, say, the smallest fill-in) and eliminate one of them.
3. **random next step:** Create a distribution over those m-best vertices, where the probability formed by either: 1) uniform, or 2) inversely proportional to the greedy score (e.g., inverse fill-in). Draw from this distribution to choose node to eliminate.

Better Heuristics for elimination

Variants and improvements to the above heuristics.

1. **tie-breaking:** When one heuristic has tie, choose one of the other heuristics to break tie.
2. **non-greedy:** Rather than greedily choosing best vertex, take the m-best vertices (e.g., the $m < n$ nodes that would result in, say, the smallest fill-in) and eliminate one of them.
3. **random next step:** Create a distribution over those m-best vertices, where the probability formed by either: 1) uniform, or 2) inversely proportional to the greedy score (e.g., inverse fill-in). Draw from this distribution to choose node to eliminate.
4. **random repeats:** Run above heuristics multiple times, producing different elimination orders. Choose one that results in the smallest maximum clique size.

k-trees and embeddings

- Goal: We want to find the elimination order that results in the smallest k such that $G' = (V, E \cup F_\sigma)$ can be embedded into a k-tree, i.e.,find best "Chordal cover"

Theorem 4.5.17

For an arbitrary graph $G = (V, E)$, finding the smallest k such that G can be embedded into a k-tree is an NP-complete optimization problem (i.e., the decision version of the problem, asking if G can be embedded into a k-tree of size k, is NP-complete).

- consider again elimination as summing out variables - not possible to guarantee optimal summation in poly-time order unless P=NP.
- We resort to heuristics (min fill, min size, random chose from top ℓ with random restarts, etc. work well).
- Inapproximability result: (see below)

Other views of the difficulty

There are a class of related problems that equivalently indicate the difficulty were are in.

Theorem 4.6.1

Given an arbitrary graph $G = (V, E)$, find the largest clique $C \subseteq V(G)$, where large is measured in terms of $|C|$ is an NP-complete optimization problem.

- Approximation algorithms - possible to do no worse than $O((\log |V|)^2/|V|)$ times size of true maximum size clique.
- Inapproximable $|V|^{1/2-\epsilon}$ for any $\epsilon > 0$.
- If we could find the smallest k such that it could be embedded it a k tree, we could identify the maximum clique in the graph. How?

Another view of the difficulty

While we're at it, even finding best chordal fill-in is hard

Theorem 4.6.2

Given an arbitrary graph $G = (V, E)$, and $G' = (V, E \cup F)$ is a triangulation of G, finding the smallest such F is an NP-complete optimization problem.

Another view of the difficulty

While we're at it, even finding best chordal fill-in is hard

Theorem 4.6.2

Given an arbitrary graph $G = (V, E)$, and $G' = (V, E \cup F)$ is a triangulation of G, finding the smallest such F is an NP-complete optimization problem.

Thus, to summarize, finding the optimal elimination order is likely computationally hard, as are other problems associated with graphs.

Some good news ☺- at least we can identify triangulated graphs

- We know that if there is a perfect elim order, the graph is triangulated.

Some good news ☺- at least we can identify triangulated graphs

- We know that if there is a perfect elim order, the graph is triangulated.
- keep eliminating simplicial nodes as long as you can, and output "not triangulated" if ever there is no simplicial node.

Some good news ☺- at least we can identify triangulated graphs

- We know that if there is a perfect elim order, the graph is triangulated.
- keep eliminating simplicial nodes as long as you can, and output "not triangulated" if ever there is no simplicial node.
- naïve implementation: find fill in of each node, eliminate the one with no fill-in $O(n^3)$.

Some good news ☺- at least we can identify triangulated graphs

- We know that if there is a perfect elim order, the graph is triangulated.
- keep eliminating simplicial nodes as long as you can, and output "not triangulated" if ever there is no simplicial node.
- naïve implementation: find fill in of each node, eliminate the one with no fill-in $O(n^3)$.
- There is a smart algorithm, maximum cardinality search (MCS), that can do this in $O(|V| + |E|)$

Some good news ☺- at least we can identify triangulated graphs

- We know that if there is a perfect elim order, the graph is triangulated.
- keep eliminating simplicial nodes as long as you can, and output "not triangulated" if ever there is no simplicial node.
- naïve implementation: find fill in of each node, eliminate the one with no fill-in $O(n^3)$.
- There is a smart algorithm, maximum cardinality search (MCS), that can do this in $O(|V| + |E|)$
- Basic idea of MCS: produce a perfect elimination order, if it exists, in reverse. Construct it by looking at previously labeled neighbors.

MCS

Input: An undirected graph $G = (V, E)$ with $n = |V|$.
Result: triangulated or not, MCS ordering $\sigma = (v_1, \dots, v_n)$

```
L ← ∅ ; i ← 1 ;
while |V \ L| > 0 do
    Choose v_i ∈ argmax_{u∈V\L} |δ(u) ∩ L| ;   /* v_i's previously labeled neighbors has max cardinality. */
    c_i ← δ(v_i) ∩ L ;   /* c_i is v_i's neighbors in the reverse elimination order. */
    if {v_i} ∪ c_i is not complete in G then
        return "not triangulated" ;
    L ← L ∪ {v_i} i ← i + 1 ;
return "triangulated", the node ordering σ
```

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 4.6.3

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 4.6.4

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

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.

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.
- Triangulation heuristics: min-fill, 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.
- Triangulation heuristics: min-fill, etc.
- MCS can identify a triangulated graph efficiently.

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.
- Tree (1-tree) case - messages for one query used for other queries. Message re-use 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.
- Tree (1-tree) case - messages for one query used for other queries. Message re-use 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.
- Tree (1-tree) case - messages for one query used for other queries. Message re-use 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 $O(r^{|c'|})$, 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 $O(r^{|c'|})$, 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 $O(r^{|c'|})$, 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.

clique queries

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

clique queries

- 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.

clique queries

- 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.

clique queries

- 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 4.7.1 (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.

$$p(x) = p(x_A, x_B, x_C) = \frac{p(x_A, x_C)p(x_B, x_C)}{p(x_C)} \tag{4.1}$$

Decomposable models

Definition 4.7.2

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.

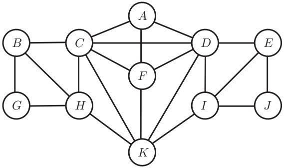
Decomposable models

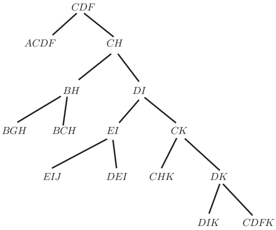
Definition 4.7.2

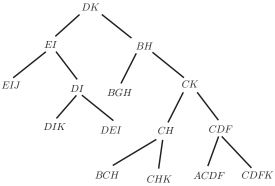
A graph $G = (V, E)$ is decomposable if either: 1) G is a clique, or 2) G possesses a proper decomposition (A, B, C) s.t. both subgraphs $G[A \cup C]$ and $G[B \cup C]$ are decomposable.

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

Decomposable models

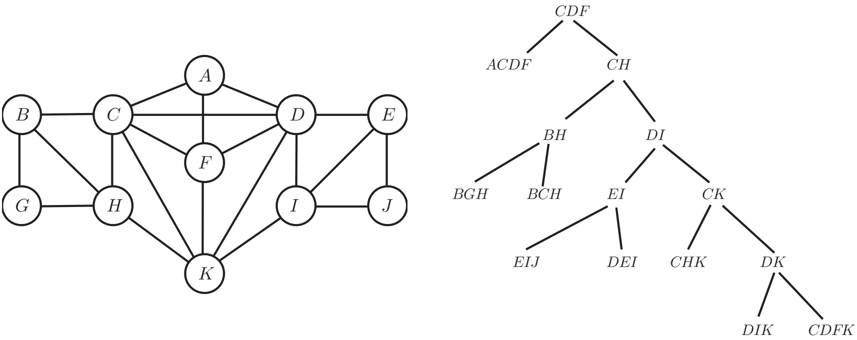






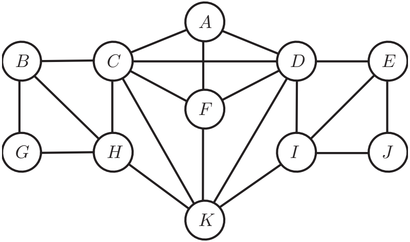
- Graph and two decompositions of this graph.
- as we recurse down, if at any point decomposition is not found, graph is not decomposable.

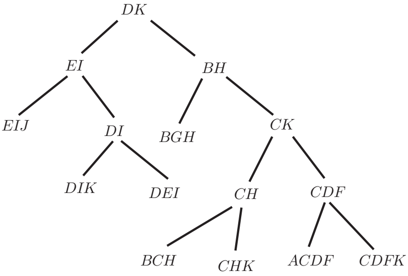
Decomposable models



- 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

Summarizing both:

Definition 4.7.3 (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 4.7.4

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

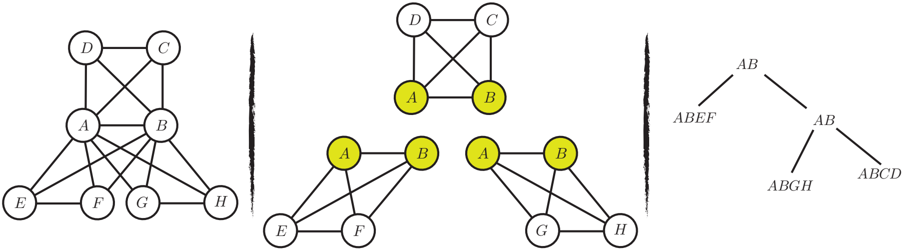
- Internal nodes in tree are complete graphs that are also separators.
- With G is decomposable, what are implications for a $p \in \mathcal{F}(G, \mathcal{M}^{(\mathrm{f})})$?

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

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

Decomposable models

- 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:

$$p(x) = \frac{\prod_{C \in \mathcal{C}(G)} p(x_C)}{\prod_{S \in \mathcal{S}(G)} p(x_S)^{d(S)-1}} \tag{4.2}$$

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

$$p(x_1, x_2, x_3, x_4) = \frac{p(x_1, x_2, x_4)p(x_1, x_3, x_4)}{p(x_1, x_4)} = \frac{p(x_1, x_2, x_3)p(x_2, x_3, x_4)}{p(x_2, x_3)} \tag{4.3}$$

Decomposable models

Proposition 4.7.5

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, 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 4.7.6

The (nec. unique) set of all minimal separators of graph are included in the non-leaf nodes of the binary decomposition tree, with $d(S) - 1$ being 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
- 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 4.7.7

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

Proof.

First: decomposability implies triangulated (induction). Next: every minimal separator complete in G implies decomposable.

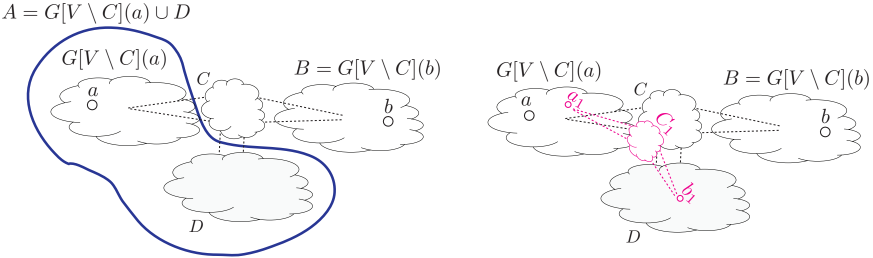
Assume G is decomposable. If G is complete then it is triangulated. If it is not complete then there exists a proper decomposition (A, B, C) into decomposable subgraphs $G[A \cup C]$ and $G[B \cup C]$ both of which have fewer vertices, meaning $|A \cup C| < |V|$ and $|B \cup C| < |V|$. By the induction 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. ...

Triangulated vs. decomposable

... proof continued.

Next, assume that all minimum (a, b) separators are complete in G. 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 $B = G[V \setminus C](b)$. 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).

Triangulated vs. decomposable



Triangulated vs. decomposable

... proof continued.

Let C_1 be a minimal (a_1, b_1) separator in $G[A \cup C]$. But then 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 regenerate G, there still cannot be any 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. Therefore, C_1 is complete in G, and an inductive argument says that $G[A \cup C]$ is decomposable. The same argument holds for $G[B \cup C]$. Therefore, G is decomposable. $\square$

Tree decomposition

Definition 4.7.8 (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

- 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

- 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 4.7.9

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

- 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 4.7.9

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.

- Multiplicatively approximable within $O(\log |V|)$, but not possible to additively do better than $|V|^{1-\epsilon}$ for any $\epsilon > 0$.

Tree decomposition is also hard

- 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 4.7.9

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.

- Multiplicatively approximable within $O(\log |V|)$, but not possible to additively do better than $|V|^{1-\epsilon}$ for any $\epsilon > 0$.
- How does this relate to our problem though?

Sources for Today's Lecture

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