Our two main texts

Announcements

- Reading assignment: Read the tree_inference.pdf chapter soon to be posted on the web page.
- Read chapters 1, 5, and 6 in the MRF book.
- At https://catalyst.uw.edu/gopost/board/bilmes/23863/ you can find a draft copy of tree_inference.pdf.

Class Road Map

We need to find one makeup lecture this term.

- L1 (9/28): Introduction, Families, Semantics
- LX (9/30): No class
- L2 (10/5): Trees, exact inference
- L3 (10/7): More on trees and inference.
- L4 (10/12): To tree or not to tree.
- L5 (10/14):
- L6 (10/19):
- L7 (10/21):
- L8 (10/26):
- L9 (10/28):
- L10 (11/2):
- L11 (11/4):
- L12 (11/9):
- LXX (11/11): Veterans Day, no class
- L13 (11/16):
- L14 (11/18):
- L15 (11/23):
- LXX (11/25): Thanksgiving, no class
- L16 (11/30):
- L17 (12/2):
- L18 (12/7):
- L19: (12/9):
Collect/Distribute Evidence

- Blue arrows indicate messages towards the root (node 1)
- Red arrow indicate messages away from the root.
- Messages abide by MPP, easy to program (simple tree traversal).

Tree queries with arbitrary $S$

- So far, we have said that $S \subset E$ so each query $p(x_S)$, $S = (i,j) \in E$.
- Ex: 4-node Markov chain: $G = x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$ with $p(x_1, x_2, x_3, x_4) \in \mathcal{F}(G, R(f))$, goal is $p(x_1, x_2, x_3)$
- We eliminate $x_4$ in

$$\sum_{x_4} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\psi_{3,4}(x_3, x_4)$$

(1)

- $O(r^2)$ computation.
- General property: if $S$ is a sub-tree in $G = (V, E)$ then can do the trick above, resulting $p(x_S)$ can be obtained by “rooting” the tree at the subtree $S$, and still have $O(r^2)$ computation.
Tree queries with arbitrary $S$

- Goal: compute $p(x_S)$
- Theorem from last time: if there are $v, w \in S$ that are connected by a path strictly within $V \setminus S$, then $v, w$ will be connected once elimination has run.
- Worst case, $S$ can become a clique, and computation will be exponential in $|S|$.
- Best case, for any $v, w \in S$ there is no path between them outside of $S$ — this is the case where $S$ induces a tree.
- Typical case: somewhere in between, depends on the query.
- Bad news for (some) scientists who want to do exact inference! 😊

---

Perfect elimination orders

**Definition 1 (perfect elimination order)**

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

- Trees always have perfect elimination orders.
- Other graphs do not.
- If we eliminate all nodes in a graph $G = (V, E)$ resulting in fill-in edges $F$, reconstitute that graph to $G' = (V, E \cup F)$, then $G'$ has a perfect elimination order (as we will see).
Non-tree graphs

- 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 $\mathcal{F}((V, E), R^{(f)}) \subset \mathcal{F}((V, E \cup F), R^{(f)})$.
- Appears to be computational equivalence classes of families of models.

Lemma 2

*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 $\sigma$ matters. Using $\sigma$ a second time results in a perfect elimination order.
**Non-tree graphs**

**Lemma 3**

*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 at 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.

**Lemma 4**

*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 \( \sigma_i \) in \( G_{i-1} \), eliminating variable \( v \) when it is in the context of its current neighbors will cost \( O(r^{\ell}) \) where \( \ell = |\delta(v) + 1| \) — 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.

...
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.

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? (HW problem)
- We can characterize the complexity of a given elimination order.
Since these are inevitable, let’s define them and give them a name.

**Definition 5 (perfect elimination graph)**

A graph $G = (V, E)$ is a perfect elimination graph if there exists an ordering $\sigma$ of the nodes such that eliminating nodes in $G$ based on $\sigma$ 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.
Maxcliques of perfect elimination graphs

Lemma 6

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 — 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 7

Given a graph $G$, an order $\sigma$, 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.
Corollary 8

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

- A node can be a member of more than 1 max-clique. 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 $p \in \mathcal{F}((V, E), R^{(f)})$. We must work with $\mathcal{F}((V, E \cup F_{\sigma}), R^{(f)})$.
- 1) Can we identify the smallest such larger family (best elimination order $\sigma$) in which inference is solved?
- 2) Exists property (other than having a perfect elimination order) that characterizes this family of graphs?

Definition 9 (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 $G_{\sigma}$.
- 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?
- 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 $p \in \mathcal{F}(G, R^{(f)})$, find some spanning subgraph $G' = (V, E')$ where $E' \subseteq E$, and solve inference there for a $p' \in \mathcal{F}(G', R^{(f)})$ 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.

![A chord (bold blue) with the cycle (bold black)](image)

### Definition 10 (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 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).
Lemma 11 (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.
Nodes that have no fill-in have a special name

Definition 12 (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.
Theorem 13

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

- TL: 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.
- TM: \{x_3, x_4\} no longer a separator. \{x_2, x_3, x_4\} now a minimal \(x_1, x_5\)-separator.
- TR: \{x_2, x_4, x_6\} minimal \(x_1, x_3\)-separator, \{x_4, x_6\} is a minimal \(x_5, x_7\)-separator.
- BL: A, F, K is a minimal \(B, E\)-separator.
- BM: D, F, K is a non-minimal \(B, E\)-separator.

Lemma 14

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 (i.e., \((G_A, G_B) = G[V \setminus S]\)) 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 15

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 $(v, w)$-separator, which is complete, so $v$ and $w$ are connected thereby creating a chord in the cycle. So all cycles are chorded. ...

... 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 $j \in \{1, \ldots, n\}$ (see figure). 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 $b_j$'s. Also, no $a_i$ is adjacent to any $b_j$ 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 16**

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?

**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 $G$. 

---

**Prof. Jeff Bilmes**

EE12A/Fall 2011/Graphical Models – Lecture 4 - October 12th, 2011
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 17**

*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:

**Lemma 18**

*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 chorded 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 19**

*A graph $G$ is triangulated iff there exists a perfect elimination order over the nodes in $G$.*
Corollary 20

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

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

**Algorithm 1:** Regenerate triangulated graph.

**Input:** A triangulated graph $G = (V, E)$ and a perfect elimination order $\sigma$.

**Result:** A new graph $G'$ identical to $G$.

1. Recall that $\delta_{G_{i-1}}(\sigma_i)$ are neighbors of $G$ at the point $\sigma_i$ is eliminated.
2. Add $\sigma_N$ to $V(G')$;
3. for $i = N - 1 \ldots 1$ do
   4. Add $\sigma_i$ to $V(G')$;
   5. Add $\delta_{G_{i-1}}(\sigma_i)$ to $E(G')$; /* at this $\delta_{G_{i-1}}(\sigma_i)$ 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?)

**Theorem 21**

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!
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?

\textbf{\(k\)-trees}

Generalizations of a tree as defined as follows:

\begin{definition} \((k\text{-tree})\)

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.

\end{definition}

- 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
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 23**

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.

**Definition 24 (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 $k$?
**Lemma 25**

*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 $\sigma = (\sigma_1, \ldots, \sigma_n)$ 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 $\{\sigma_{n-k}, \sigma_{n-k+1}, \ldots, \nu_n\}$ 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 $\sigma_i$ is adjacent to a clique $c$ with no more than $k$ vertices in $\{\sigma_{i+1}, \ldots, \nu_n\}$. In the $k$-tree $T_{i+1}$, $c$ is contained in a $k$-clique $c'$. When we make $v_i$ adjacent to all of the vertices of $c'$, we obtain a $k$-tree $T_i$ since $v_i$ is still simplicial in $T_i$. Repeating to $v_1$ and result is supergraph of $G$ with same order being perfect.
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.

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:

---

**Theorem 26**

*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).*

this means that we can’t expect to find perfect elimination order.

if we already know \( k \), we can find optimal \( k \)-tree cover in polynomial time but it is expensive.
Heuristics for elimination

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

Note that there are a number of variants and improvements to the above three.

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.
Scratch Paper
Sources for Today’s Lecture

- Most of this material comes from the tree_inference.pdf handout posted at https://catalyst.uw.edu/gopost/board/bilmes/23863/.