11.1 Graph Theory (review from last time)

Theorem:
(decomposable) \equiv (chordal) \equiv (every min. \alpha - \beta-separation is complete) \equiv (there exists a junction tree of cliques)

We want to try to get any graph into this family of equivalences, so that we can then construct a junction tree from the graph.

Next time we will relate this to independence and inference.

Running Intersection Property (RIP): a sequence (C_1, C_2, ..., C_n) of sets has the RIP if
\forall j, S_j \subseteq C_i for some i < j, where
S_j = H_{j-1} \cap C_j (the separator set)
H_j = C_1 \cup C_2 \cup ... \cup C_j (the history up to and including j)
R_j = C_j \setminus H_{j-1} (the residual or innovation)

Theorem:
The sequence of cliques formed by a well-ordering (e.g., breadth-first or depth-first search order) of a junction tree has the RIP

Why is this true? S_j = H_{j-1} \cap C_j = C_{p_j} \cap C_j \Rightarrow S_j \subseteq C_{p_j} (since it is a junction tree), where P_j < j, a “neighbor” in the tree according to the search order

11.2 Constructing a Junction Tree

Algorithm: constructing a junction tree
input: C_1, C_2, ..., C_p (the cliques of a chorded graph satisfying RIP)
step 1: associate each node in the graph with each clique
step 2: for j = 2, ..., p
add an edge between C_i and C_j for some i < j such that S_j \subseteq C_i (S_j = H_{j-1} \cap C_j)

Intuition: Why does it work?
For a junction tree, for any node v, cliques containing v must induce a connected tree, and there are two cases:
Case 1: v \in S_j, and we know that S_j \subseteq C_i and S_j \subseteq C_j, which means that v \in C_i and v \in C_j

Case 2: v \not\in S_j, which means that v \in R_j = C_j \setminus H_{j-1}
Note: node \( j \) is a candidate for some later node that has \( v \).

See Lauritzen for the formal proof.

### 11.3 More Graph Theory Basics

*Definition: Perfect DAG* - a DAG is perfect if the parents of every node form a complete set.

![A Perfect DAG](image1)

![NOT a Perfect DAG](image2)

*Figure 11.1: Examples of perfect and non-perfect DAGs*

*Definition: Perfect Numbering* - a node numbering \((v_1, v_2, ..., v_n)\) of an UGM is perfect if the neighbors of any node that have a lower number than that node are complete, that is for which \( \text{ne}(v_j) \cap \{v_1, v_2, ..., v_n\} \) is complete.

![Example Graph: ABCDEF is a perfect numbering but ABCEFD is not.](image3)

Is ABCDEF a perfect numbering? yes
Is ABCEFD a perfect numbering? no

Note: any ordering of a complete graph is perfect

*Theorem:*

Give a well-ordered (e.g., topological order) \((v_1, v_2, ..., v_e)\) perfect DAG \( G \), its undirected version \( G^\sim \) is chordal with \( v_1, v_2, ..., v_e \) a perfect numbering

Why? \( G^\sim \) is decomposable: \( \forall j, (W_j, V_{j-1}, S_j) \) form a decomposition, where:

- \( W_j = d^\sim(V_j) \cap V_{j-1} \) (note that \( d^\sim \) denotes closure w.r.t. the undirected graph)
- \( S_j = W_j \cap V_{j-1} \)
- \( V_j = \{v_1, v_2, ..., v_j\} \)
(W_j, V_{j-1}, S_j) = (cl(V_j) \cap \{v_1, v_2, ..., v_j\}, \{v_1, v_2, ..., v_{j-1}\}, cl(V_j) \cap \{v_1, v_2, ..., v_j\})

The first term $cl(V_j) \cap \{v_1, v_2, ..., v_j\}$ is complete and therefore decomposable.
The second term $\{v_1, v_2, ..., v_{j-1}\}$ is decomposable by induction.
The third term $cl(V_j) \cap \{v_1, v_2, ..., v_j\}$ is complete.

Suppose we are given a perfect numbering $(v_1, v_2, ..., v_j)$ of an UGM. We can produce a perfect DAG by pointing lower numbered vertices to higher numbered vertices.

![Figure 11.3: Producing a perfect DAG from an UGM.](image)

**Theorem:** (again from Lauritzen)
An UGM is chordal iff it admits a perfect numbering.

Summary of equivalent conditions on graphs:

- decomposable
- chordal
- every minimal $(\alpha, \beta)$-separator is complete.
- there is a junction tree of cliques
- the graph admits a perfect numbering

But how do we tell if a graph is chordal? Two important definitions:

**Definition:** simplicial - $\alpha$ is simplicial if $bd(\alpha)$ is complete (all neighbors are joined in UGM)

![Figure 11.4: Examples of simplicial and non-simplicial nodes.](image)

**Definition:** eliminatable - a graph $G$ is eliminatable if all nodes can be successfully removed by the elimination algorithm without adding any edges

Remember elimination algorithm (from first 2 lectures):
1. choose an ordering
2. pick nodes in order, join all neighbors
3. remove node, repeat

Note: running elimination on a perfect numbering produces a chordal graph

**Theorem:**
Any eliminatable graph is chordal.

**Proof:** by induction
Base case: 1 node is obvious
Assume true for N, show N+1
Given an eliminatable graph with N+1 nodes, we know that it has at least one simplicial node A
\( G \setminus \{A\} \) is triangulated by induction (i.e. since \( G \setminus \{A\} \) is eliminatable also)
but A doesn’t produce any chordless cycles because it is simplicial
\( \Rightarrow G \) is chordal

But how do we know if a graph is eliminatable? Could we run the elimination algorithm many times for all possible orderings? No, this would be prohibitively expensive. We need an efficient procedure for testing the chordality of a graph.

### 11.4 Maximum Cardinality Search

Maximum Cardinality Search (MCS) is an algorithm that tests an UGM for chordality in \( O(V+E) \) time. First published by Tarjan and Yannakakis (1984), it is a very important algorithm for Bayes Net systems.

Full reference:
**Algorithm**: Maximum Cardinality Search

**Input**: undirected graph \( G \)

**Outputs**: 1) whether \( G \) is chordal or not  
2) a perfect numbering for \( G \), if it is chordal

set output = “chordal”

set \( i = 1 \) ;; \( i \) is a counter

set \( L = \emptyset \) ;; \( L \) is the set of previously labeled vertices

set \( c(v) = 0 \forall v \in V \) ;; array counting number of previously labeled neighbors of each node \( v \)

while (\( L \neq V \))

\( U = V \setminus L \) ;; current set of unlabeled vertices

\( v^* = \arg\max_{v \in U} c(v) \) ;; find, from the unlabeled nodes, one with the greatest number of previously labeled neighbors, breaking ties arbitrarily.

\( v_i = v^* \) ;; rename so \( v^* \) gets the name \( v_i \)

if \( ne(v_i) \cap L \) is not complete ;; bors is not complete, not chordal

output = “not chordal”

\( \pi_{v_i} = ne(v_i) \cap L \) ;; set of prev. labeled neighbors of \( v_i \)

\( c(w) = c(w) + 1, \forall w \in (ne(v_i) \cap U) \) ;; increment counts of all neighbors of this newly labeled node. Later we pick one with many neighbors.

\( L = L \cup \{v_i\} \)

\( i = i + 1 \)

The idea is that we maximize the cardinality of the set of previously labeled neighbors, i.e., \( ne(v_i) \setminus L \). The cardinality is increased by increasing \( c(w) \), so next node we choose will have many labeled neighbors. We can prove that if ever such a set is not complete, then it is not chordal.
Note: the first node in the ordering is chosen arbitrarily, since \( v^* = 0 \) for all nodes during the first iteration.

Also note: the resulting labels \( (v_1, v_2, \ldots, v_k) \) constitute a vertex ordering. This ordering will be perfect (if the graph is chordal) since for any node \( v, \pi_v \) will be complete.

### 11.5 Finding Cliques Satisfying RIP

Key idea: given the ordering produced by MCS, we can find a clique ordering satisfying RIP

<table>
<thead>
<tr>
<th>Algorithm:</th>
<th>ladder-based clique construction</th>
</tr>
</thead>
<tbody>
<tr>
<td>start with node ordering ( (v_1, v_2, \ldots, v_k) ) from MCS</td>
<td></td>
</tr>
<tr>
<td>( \pi = \pi_{v_i} ) ( : ) cardinality of sets of previously labeled neighbors</td>
<td></td>
</tr>
<tr>
<td>define ( i ) as “ladder node” if ( \pi_i \geq \pi_{i+1} OR_i = k )</td>
<td></td>
</tr>
<tr>
<td>( \lambda_j = j ) th ladder node</td>
<td></td>
</tr>
<tr>
<td>( c_j = {\lambda_j} \cup \pi_{\lambda_i} )</td>
<td></td>
</tr>
</tbody>
</table>

**Theorem:**

\( c_j \) are the cliques of \( G \) and clique ordering \( (c_1, c_2, ...) \) satisfies the RIP

**Proof:**

see Cowel et al. “Probabilistic Networks and Expert Systems”

Table 11.1 shows the result of running the ladder-based clique construction algorithm on the node ordering produced by the MCS algorithm.

The clique ordering produced by this algorithm is: \{3,1,2\}, \{5,2,3\}, \{6,2,5\}, \{4,2\}.
Table 11.1: Ladder-based Clique Construction.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>6</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_{v_i}$</td>
<td></td>
<td>0</td>
<td>{1}</td>
<td>{1,2}</td>
<td>{2,3}</td>
<td>{2,5}</td>
</tr>
<tr>
<td>$\pi_i$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Ladder?</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

11.6 Look-ahead Triangulation

What if the graph is not chordal or if it is a DAG? We need the smallest clique sizes in order to make probabilistic inference efficient, but choosing the best triangulation is an NP-Hard problem.


**Algorithm:** One-step look-ahead triangulation

- set $i = k$ ($k$ = number of nodes)
- unnumber all nodes
- while (exists unnumbered nodes)
  - select unnumbered node $v$ that optimizes $c(v)$
  - label $v$ as $i$
  - form set $C_i$ consisting of $v_i$ and its unnumbered neighbors
  - join all unjoined pairs of nodes in $C_i$
  - eliminate $v_i$
  - $i = i + 1$

Note: quality of triangulation depends on choice of $c(v)$

**Theorem:**
Elimination yields a triangulated graph

**Proof.** (by induction)
- single nodes are obvious, so assume holds for $N$, show for $N + 1$.
- Eliminating a node results in $N$ node graph which is chorded by induction.
- The elimination step by itself does not produce chordless cycles since it joins all neighbors of nodes (forming a complete set). Any chordless cycle must go through that node.

Note: not every tree of cliques from a chorded graph is a junction tree, as we will see. The weight of a tree of cliques $w(t) =$ sum of cardinalities of separator sets needs to be maximum.