14.1 Review of Last Lecture

Last time we proved the important theorem that states that the following properties of a graph are all equivalent:

- Decomposable
- Chordal (triangulated)
- Every minimum \((\alpha, \beta)\) separator is complete
- There exists a junction tree of cliques of the graph
- Eliminatable (i.e. there exists an ordering during elimination that does not add edges)
- The exists a perfect numbering of the graph

We have also proved that the sequence of cliques formed by a well-ordering of a junction tree has the running intersection property (RIP).

We defined simplicial nodes: \(\alpha\) is simplicial if \(ne(\alpha)\) is complete and we used the definition in the proof that any eliminatable graph is chordal.

Finally we described maximum cardinality search and showed how it tests chordality of an undirected graph and yields a an elimination ordering if the graph is chordal.

What do we do if the the graph is not chordal, however? We need to triangulate the graph. In the next section we will look at an algorithm do achieve that.

14.2 Look-ahead Triangulation

In order to do efficient inference on the graph, we want small cliques in the triangulated graph. Unfortunately, finding the best triangulation of a graph is an NP-hard problem.


The following one-step look-ahead triangulation is a heuristic that triangulates a graph.
Algorithm: One-step look-ahead triangulation (Elimination++)

Input: array \(c(v)\), cost of adding node \(v\)

set \(i=k\) (\(k\) = number of nodes)

unnumber all nodes

while (∃ unnumbered nodes)
    select next unnumbered node \(v\) that minimizes \(c(v)\)
    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: The quality of the triangulation depends on choice of \(c(v)\).

Theorem:
Elimination++ yields a triangulated graph. (*)

Proof: (by induction)
Base case: \(|v| = 1\). Trivially true.

Assume (*) holds for \(N\) or fewer nodes and show for \(N+1\).

We have the following:

- Eliminating a node results in \(N\) node graph, which is chorded by the induction hypothesis.
- The elimination step adds no chordless cycles since it joins all neighbors (forming a complete set). Any cycle must go through those neighbors.

Therefore, (*) is true for \(N+1\) nodes, and hence the theorem holds.

More generally we have the following very important theorem which states that a graph is triangulated if and only if there exists a perfect order. This means that to test if a graph is triangulated, it is sufficient to see if the graph is eliminatable, meaning if we can find a perfect ordering. Often this is an easier thing to do than to see if all the cycles of the graph have a chord.

**Theorem 14.1.** A graph \(G\) is triangulated iff there exists a perfect order (a perfect elimination order is one that adds no edge during elimination).


**Corollary 14.2.** Elimination is sufficient: all triangulations can be obtained with appropriate elimination orders.

Note: If \(|v| = n\), there are \(n!\) possible triangulations, which means that it is too costly to go though all possible triangulations and find the best one. We’ll need heuristics.
14.2.1 Optimization criteria

In what follows we consider three simple heuristics that produce elimination orders according to different notions of what we might want in an optimal triangulation. Note that these schemes are heuristics and none of them are optimal, but in practice they have shown to produce remarkably good triangulations and are often just as good as much more elaborate heuristics.

1. Minimize fill-in during elimination (minimum deficiency): as we eliminate, find the ordering that adds as few edges as possible. This means that at each stage when we are choosing the next node to eliminate, we choose the node (amongst all the nodes which are candidates to eliminate at the current time) which produces the fewest number of additional edges.

2. Minimum degree heuristic (also called min size in Kjaerulf below). Here, at each stage in the elimination, we choose the next node that has minimal degree (i.e., we choose the node that, among all other candidates at the current time, has the fewest incident edges).

3. In this last scheme, we attempt to produce a graph that has minimal weight, where the minimal weight is defined as 
\[ w(G) = \sum_{C \in C} \prod_{v \in C} s(v) \] 
where \( s(v) \) is the cardinality of \( v \) (the number of different values it can take). Note that the weight of a node is defined as 
\[ w(g) = \prod_{w \in \text{ne}(v)} s(v) \], and the heuristic says that we choose during elimination which has minimal weight. Note that this scheme is identical to the minimal degree heuristic when all nodes have the same cardinality (i.e., \( s(v) = s(w) \) for all \( v \) and \( w \)).

To do efficient inference, we are most often interested in the third optimization criterion because the complexity of inference is equal to the sum of the clique state space sizes.

Example: Complexity of HMM inference

![Figure 14.1: Cliques in an HMM](image)

If the cardinality of a hidden node is \( N \) and the chain is \( T \) long, the weight, \( w(G) \) of the HMM is then 
\[ w(G) = T \times (N^2 + N) \], where \( N^2 \) is the state space of the “horizontal” cliques and \( N \) the state space of the “vertical” cliques. The cost of inference is then \( O(TN^2) \).

14.2.2 Other triangulation schemes

There are many triangulation schemes, some of which are listed below:

- Kjaerulff, “Triangulation of Graphs” 1990 Uses the following techniques:
  - Given a fill-in \( T \), try to find \( T' \in T \) such that the graph is still triangulated.
  - Simulated annealing approach.

  This paper also talks about the three heuristics above, and finds that they all do pretty well most of the time. The simulated annealing approach is much more complex to run, and only sometimes produces...
better triangulations. Also see the paper ‘A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations’ in Graph Theory and Computing, R. Read ed. Academic Press, NY 1973

- Meila and Jordan 1996
- Beaker and Geiger 1996

### 14.3 Kruskal’s Algorithm for finding Junction Trees

Note: Not every tree of cliques from a chorded graph is a junction tree.

![Not a Junction Tree vs. Junction Tree](image)

Figure 14.2: Not every tree of cliques is a Junction Tree

Notice that the separator sets are larger in the Junction Tree than in the non-Junction Tree.

**Definition 14.3.** The **Weight**, \( w(t) \), of a tree of cliques \( \equiv \) the sum of the cardinalities of the separator sets.

**Theorem 14.4.** A tree of cliques is a Junction Tree if it is the maximal spanning tree on the graph of cliques with edge weights set according to the cardinality of the separator sets. The graph of cliques is constructed by connecting any cliques which have at least one node in common.

We can form the maximal spanning tree by using Kruskal’s algorithm.

#### 14.3.1 Kruskal’s Greedy Algorithm for finding a maximal spanning tree

This algorithm will find the maximal spanning tree of a graph

1. Start with no edges in the tree
2. choose the edge with the largest weight which doesn’t produce any cycles
3. repeat the above step until the graph is fully connected

The proof of correctness for the algorithm can be found in most Algorithms books. One good one is Cormen, Leiserson, and Rivest “Intro to Algorithms”
Figure 14.3: Example of a maximum spanning tree on a graph. The heavy lines are those which would be included in the tree, given the edge weights shown, if Kruskal’s Algorithm for finding the maximal spanning tree were run on the graph.

**Theorem 14.5.** The maximal spanning tree on a graph of cliques is a Junction Tree

Consider a tree of M cliques, named $C_i$ with separators $S_j$. Consider $x_k$.

- number of times $x_k$ appears in a separator is $\sum_{j=1}^{M-1} 1_{\{x_k \in S_j\}}$
- number of times $x_k$ appears in a clique is $\sum_{i=1}^{M} 1_{\{x_k \in C_j\}}$

where $1_{\{A\}}$ is an indicator function that has the value 1 when the event $A$ is true, and is 0 otherwise.

Figure 14.4: Section of a Junction tree under consideration

$\forall x_k, x_k \in S_j$ implies $x_k \in C_l$ and $x_k \in C_m$

Therefore, $x_k$ will appear at least one time more often in a clique than in a separator ad we have the following inequality:

$$\sum_{j=1}^{M-1} 1_{\{x_k \in S_j\}} \leq \sum_{j=1}^{M} 1_{\{x_k \in C_j\}} - 1$$

Equality holds iff the subgraph induced by $x_k$ is a tree because in a connected tree the number of cliques is exactly the number of separators plus one.

Consider starting the construction of the tree where $x_k$ is in two connected cliques, and thus in one separator. In order to maintain this difference of only one, every time we add another clique containing $x_k$, we must attach it to one of the already existing cliques that contain $x_k$. Otherwise, we have increased the number of times $x_k$ is in a clique, but not the number of times $x_k$ is in a separator, and thus we will no longer get equality in the above equation, but this criterion of maintaining equality essentially carves out the JT property.

Since the subgraph induced by $x_k$ is a tree, for all $x_k$, it must be a Junction Tree.

**Theorem 14.6.** A tree of cliques is a Junction tree iff it is a maximal spanning tree on the graph of cliques
Proof. Weight of Tree

\[
W(T) = \sum_{j=1}^{M-1} |S_j| = \sum_{j=1}^{M-1} \sum_{k=1}^{N} 1_{\{x_k \in S_j\}} \quad \text{(N=number of nodes)} \tag{14.1}
\]

\[
= \sum_{k=1}^{N} \sum_{j=1}^{M-1} 1_{\{x_k \in S_j\}} \tag{14.2}
\]

\[
\leq \sum_{k=1}^{N} \left[ \sum_{j=1}^{M} 1_{\{x_k \in C_j\}} - 1 \right] \tag{14.3}
\]

\[
= \left[ \sum_{j=1}^{M} \sum_{k=1}^{N} 1_{\{x_k \in C_j\}} \right] - N \tag{14.4}
\]

\[
= \left[ \sum_{j=1}^{M} |C_j| \right] - N \tag{14.5}
\]

Which is maximized when the inequality 14.3 is equal, which is when it is a Junction Tree

\[\square\]