Lecture 16:

Junction Trees

Sam Roweis

March 3, 2004

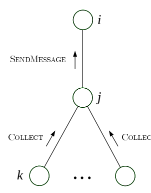
Belief Propagation (Sum-Product) Algorithm

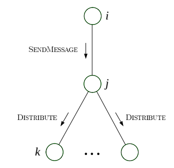
- Choose a root node arbitrarily.
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root and then back down using:

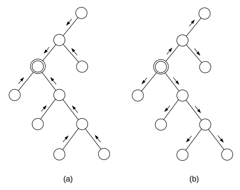
$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

- Compute node marginals using the product of incoming messages:

$$p(x_i|\bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$





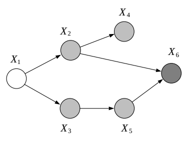


Efficient Probabilistic Inference

- We want to efficiently compute the posterior $p(\mathbf{x}_F|\mathbf{x}_E)$ over *query nodes* $\mathbf{x}_F$, conditioned on *evidence nodes* $\mathbf{x}_E$ while *integrating out marginal nodes* $\mathbf{x}_R$ when the joint distribution is represented by a directed or undirected graphical model.
- For some models (e.g. mixtures and factor analysis), this was an easy application of Bayes rule. For a single node posterior (i.e. $\mathbf{x}_F$ is a single node), there was a simple elimination algorithm. But it required a *node ordering* to be given, which told it which order to do the summations in and work could not be reused.
- We seek a general algorithm that is correct and takes advantage of the Markov properties of the graphical model to decompose a general calculation into a linked set of local computations.
- For tree-structured graphical models, there was the efficient Belief Propagation algorithm, but it didn't work on general DAGs.

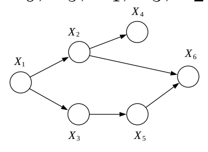
Junction Trees: Beyond Trees

- What can we do if the underlying graph is not a tree?
- Belief Propagation was efficient in its data structures and reuse of message computations, but it only works on trees.
- Eliminate worked on all directed acyclic graphs, but it was inefficient and depended on the query (elimination ordering).
- Can we get the best of both worlds and discover a "tree-like" data structure and message passing system to do efficient inference even when the underlying graph is not a tree?
- Yes, using a family of algorithms called *junction tree* algorithms.
- Let's go back and see what made Eliminate work...



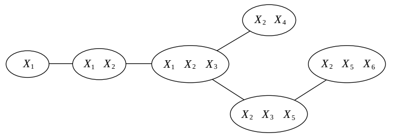
Elimination Cliques

- During a run of Elimination, we remove nodes by summing them out and then couple their remaining neighbours.
- Sets of nodes that get coupled at any point during the run are called *elimination cliques*.
  e.g. for the order $x_6, x_5, x_4, x_3, x_2, x_1$ we have:



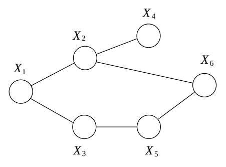
$\{X_2, X_5, X_6\}$
$\{X_2, X_3, X_5\}$
$\{X_2, X_4\}$
$\{X_1, X_2, X_3\}$
$\{X_1, X_2\}$
$\{X_1\}$

(a) (b)



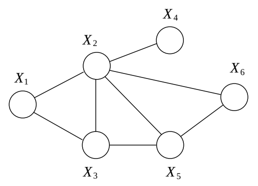
(c)

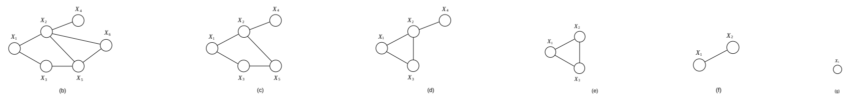
Added Edges == Triangulation



(a)

$\Rightarrow$





Triangulated Graph == undirected graph with no "chordless cycles"
a "chordless cycle" has no edges between nodes that are not sucessors

Node Elimination

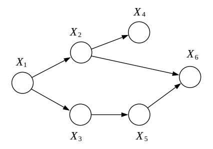
- We eliminate nodes from the graph one by one.
  For directed graphs we moralize and drop edge directions.
  For undirected graphs:

```
foreach node $x_i$ in ordering $I$:
    connect all the neighbours of $x_i$
    remove $x_i$ from the graph
end
```

- The removal operation requires summing out x_i.
- Summing out x_i leaves a function involving all its previous neighbours and thus they become connected by this step.
- If we imagine adding an edge to the graph every time a run of Eliminate occurs, the original graph becomes augmented.
- The original graph, augmented by all the added edges is now a *triangulated* graph. These added edges are why Eliminate works.

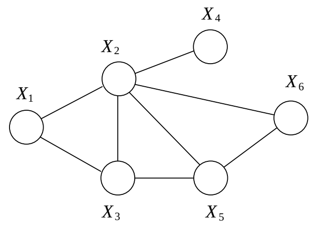
Elimination Cliques are Cliques of Triangulated Graph

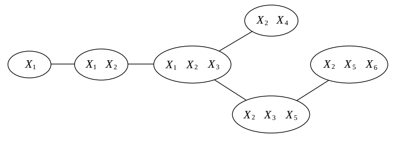
- The elimination cliques created during a run are exactly the maximal cliques of the (moralized) triangulated version of the original graph.



$\{X_2, X_5, X_6\}$
$\{X_2, X_3, X_5\}$
$\{X_2, X_4\}$
$\{X_1, X_2, X_3\}$
$\{X_1, X_2\}$
$\{X_1\}$

(a) (b)





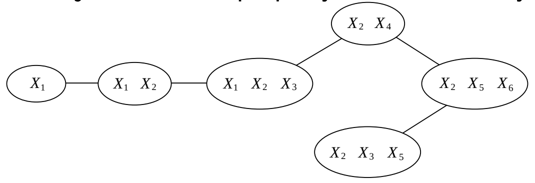
(c)

Elimination Clique Trees

- As we sum out or condition on variables, we need to pass information between these storage sites (elimination cliques).
- Elimination essentialy creates a tree of cliques during each run based on the elimination ordering. Each ordering may create a different clique tree, and may triangulate the graph differently.
- Information flows only along one path: from the clique created by eliminating the first variable in the order to the last.
- Our goal is to create a more flexible "hypergraph" connecting the cliques so that information can flow in many ways.
- In fact, this is what belief propagation (sum-product) does. It created elimination clique trees in which information could flow in both directions across the links. But for trees, there are no undirected cycles, so all elimination orderings produced the same unique clique tree.

Junction Trees

- Idea: triagulate non-tree graphs to get a set of cliques and arrange these cliques into a tree. Then do BP on this tree.
- But not all clique trees are equal! If, for every variable, all occurances of the variable appear in a connected subtree, then the clique tree has the *junction tree* property, and this is very desirable.

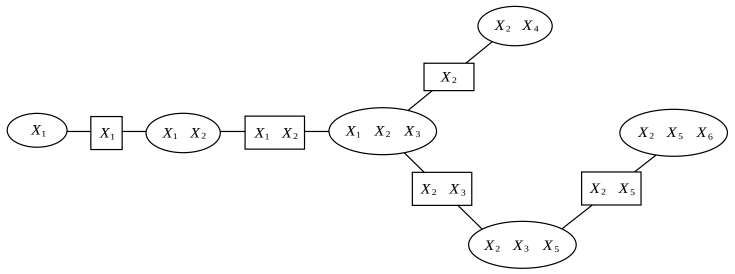


e.g. this clique tree does not have the j-tree property (x_3)

- Imagine arranging the elimination cliques into a tree structure.
  Q: Which structure should we pick for efficient calculations?
  A: One in which local information transfer is sufficient to update all necessary quantities. More on this later.

Separator Sets

- We can explicitly show what information will be communicated from clique to clique in a clique tree by taking the intersection of variables in adjacent cliques.
- These intersections are called *separator sets* and are themselves cliques (fully connected in the moralized, triangulated graph) although of course they are no longer maximal.



Potentials on Junction Trees

Here is the setup for our general inference algorithm so far:

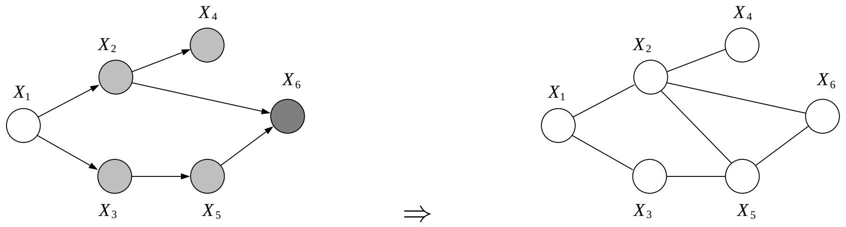
1. Start with a moralized, triangulated graph $G(V, E)$ and identify its maximal cliques C. (For triangulated graphs this is easy.)
   These correspond to the elimination cliques we had previously.
2. Arrange the cliques into a tree that has the *junction tree* property.
3. Associate a potential function with each clique in this hypergraph, and define the joint probability as the product of these potentials:
   $$p(\mathbf{X}) = \frac{1}{Z} \prod_{\text{cliques } c} \psi_c(\mathbf{x}_c) \qquad Z = \sum_{\mathbf{X}} \prod_{\text{cliques } c} \psi_c(\mathbf{x}_c)$$
   very similar to setup for undirected models, except that the cliques here are maximal, whereas in undirected graphs they might not be.
4. Initialize these potentials so that they represent the correct model. Then try to condition and marginalize to perform inference by running belief propagation on this tree.

Initializing Potentials

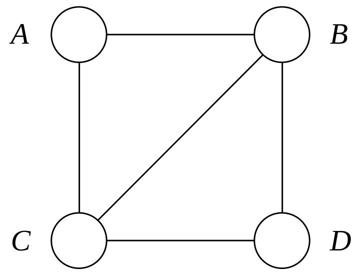
- How do we initialize the potentials of the clique tree (hypergraph) based on the original graphical model so that the distribution is correctly represented?
- Easiest case: original graphical model is undirected, triangulated and its cliques already correspond to the maximal cliques. Then we just assign potentials in a one-to-one fashion.
- Next easiest case: original model is undirected, and triangulated but with non-maximal cliques.
  1. Assign any potential completely contained within only a single maximal clique to that clique.
  2. Assign potentials that are contained by more than one clique arbitrarily to exactly one of them. Multiple ways to do this.

Initializing from Directed Graphs: Moralization

- How do we initialize from an underlying directed graph?
- Same idea, but with a slight complication: for directed graphs, the parents may not be explicitly connected, but they are involved in the same "potential" function $p(x_i|x_{\pi_i})$, (i.e. parental conditionals can have sets of arguments that are not contained in *any* maximal clique if the parents are not connected.)
- To handle this we *moralize* the graph: link parents and drop directions of edges.



Initializing from Undirected Graphs



ψ_{AB}
ψ_{AC}
ψ_{BC}
ψ_{BD}
ψ_{CD}

- Two choices:

$\psi_{ABC} = \psi_{AB}\psi_{AC}$
$\psi_{BCD} = \psi_{BC}\psi_{BD}\psi_{CD}$

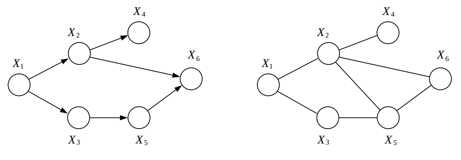
OR

$\psi_{ABC} = \psi_{AB}\psi_{AC}\psi_{BC}$
$\psi_{BCD} = \psi_{BD}\psi_{CD}$

- Each underlying undirected graph potential gets assigned to *one and only one* clique potential in the junction tree.

Moralization

- Now the conditionals are guaranteed to be functions on cliques and we proceed as above (plus these will be automatically normalized).
- Caution: moralization *adds* edges to the graph and converts it to an undirected model. So it changes the semantics.
- All original dependencies are still possible. But we may have lost some conditional independencies.
- Moralization is essential: since conditioning couples parents in directed models ("explaining away") we need a mechanism for respecting this when we do inference.



(a) (b)

Looking Ahead

Summary so far:

1. Identify maximal cliques of underlying triangulated graph. (easy)
2. Arrange cliques into junction tree.
3. Associate potentials with each clique.
4. Initialize (moralizing first if necessary) by assigning potentials or conditionals from the underlying graphical model to clique potentials in the junction tree.
5. Coming up next: introducing evidence (conditioning).
6. And then: marginalizing.

The plan from here onwards:

- We will break the general inference problem into two parts:
  1. Condition on what we observe (evidence).
  2. Then marginalize out anything we need to (sum).