Readings #5: Inference with Bayesian Networks — Clique Trees*

Daphne Koller
Stanford University

Nir Friedman
Hebrew University

In the previous chapter, we showed how efficient inference can be done in a Bayesian network using an algorithm called *Variable Elimination*, that sums out the joint one variable at a time. This algorithm is not the one used in most real systems. The algorithm that is used is called the *Clique Tree* algorithm (or *junction tree* or *join tree*). While the algorithm appears quite different, it is actually doing precisely the same operations: multiplying factors and summing out variables. We now show the clique tree algorithm and its connection to variable elimination.

1 Variable elimination as message passing

Consider again the Asia network, and recall the factors that were introduced in the different steps of the summation:

step	vareliminated	varsinfactor	resultingfactor
(1)	V	$\{V,T\}$	$f_1(T)$
(2)	X	$\{X,A\}$	$f_2(A)$
(3)	S	$\{S,L,B\}$	$f_3(L,B)$
(4)	T	$\{A,L,T\}$	$f_4(A,L)$
(5)	L	$\{A,L,B\}$	$f_5(A,B)$
(6)	A	$\{A,D,B\}$	$f_6(B,D)$
(7)	B	$\{D,B\}$	$f_7(D)$

Let's call the intermediate factors, prior to the summing out of the variable, h_i. Let's consider the data structures used in this computation. Each factor h_i needs to be stored in some table of the appropriate dimensions. For example, h_1 needs to be associated with a table with a single entry for every combination of values of V, T. To get $f_1(T)$, we simply sum out V in this data structure. Each data structure is associated with a *cluster* of variables, which is the domain of the factor.

Now, let's visualize what our computation does in terms of the clusters. We'll draw a graph whose nodes correspond to the clusters, each labelled with its domain. We'll draw an edge between two clusters if the result of the computation in one participates in the computation of the other. In other words, if we generated $f_1(T)$ in $\mathbf{C}_1$, and we used it in $\mathbf{C}_4$, we would make an edge between $\mathbf{C}_1$ and $\mathbf{C}_4$. We'll mark that edge with T, which we call the *separator*. We call $f_1(T)$ the *message* between $\mathbf{C}_1$ and $\mathbf{C}_4$. The result is, by definition, a tree. Each data structure participates only once, and transmits its information to some other data structure. We will call the resulting tree a *cluster tree*.

Definition 1.1: Let G be a Bayesian network structure over the variables $\mathbf{X}$. A *cluster tree* over G is a tree each of whose nodes is associated with a *cluster*, i.e., a subset of $\mathbf{X}$. Each edge is annotated with a subset of Bayesian network nodes called a *separator*. ∎

*This is based on draft of a text book, some of the references and citations are missing

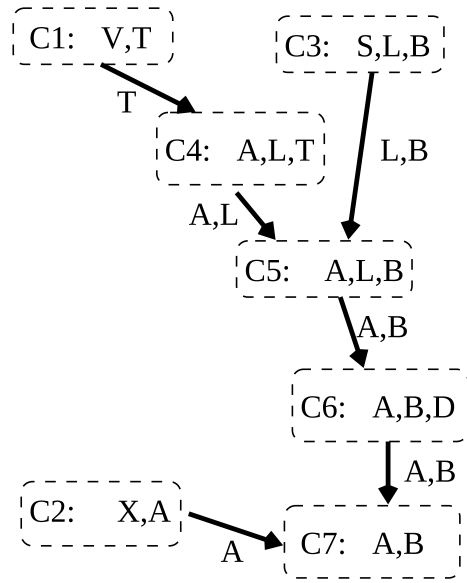

Figure 1: Cluster tree for the Asia network.

We have just shown that the variable elimination algorithm induces some particular cluster tree. The cluster tree induced by our computation over the Asia network is shown in Figure 1.

We can prove several interesting properties of this tree, which will be central later on.

Definition 1.2: Let X be a node in a Bayesian network G. We define the *family* of X to be $\mathbf{Family_X} = \{\mathbf{X}\} \cup \mathbf{Pa_X}$. We say that a cluster tree $\mathcal{T}$ over G is *faithful* to the network G if, for every X in G, there exists some cluster $\mathbf{C}$ in $\mathcal{T}$ such that $\mathbf{Family_X} \subseteq \mathbf{C}$. ∎

Proposition 1.3: *Let $\mathcal{T}$ be a cluster tree induced by a variable elimination algorithm over some Bayesian network G. Then $\mathcal{T}$ is faithful to G.*

Proof: At some point in the VE algorithm, we must multiply $P(X_i \mid \mathbf{Pa_{X_i}})$ into some factor h_j. We will then have that $X_i, \mathbf{Pa_{X_i}} \subseteq \mathbf{C_j}$. ∎

Definition 1.4: Let $\mathcal{T}$ be a cluster tree over a Bayesian network structure G. We say that $\mathcal{T}$ has the *running intersection property* if, whenever there is a variable X such that $X \in \mathbf{C}$ and $X \in \mathbf{C}'$, then X is also in every cluster in the path in $\mathcal{T}$ between $\mathbf{C}$ and $\mathbf{C}'$. ∎

It is easy to see that this property holds for our cluster tree. For example, A is present in $\mathbf{C}_4$ and in $\mathbf{C}_2$, so it's also present in $\mathbf{C}_5$, $\mathbf{C}_6$, and $\mathbf{C}_7$.

We now prove that this holds in general. Intuitively, a variable appears in every expression from the moment it is introduced (by multiplying in a factor that mentions it) until it is summed out.

Theorem 1.5: *Let $\mathcal{T}$ be a cluster tree induced by a variable elimination algorithm over some Bayesian network G. Then $\mathcal{T}$ satisfies the running intersection property.*

Proof: Let $\mathbf{C}$ and $\mathbf{C}'$ be two clusters that contain X. Let $\mathbf{C}_X$ be the cluster where X is eliminated. (If X is a query variable, we assume that it's eliminated in the last cluster.) We will prove that X

must be present in every cluster on the path between $\mathbf{C}$ and $\mathbf{C}_X$, and analogously for $\mathbf{C}'$, thereby proving the result. First, we observe that $\mathbf{C}$ cannot be "upstream" from $\mathbf{C}_X$ in the computation: When X is eliminated in $\mathbf{C}_X$, all of the factors involving X are multiplied into $\mathbf{C}_X$; the result of the summation does not have X in its domain. Hence, after this elimination, $\mathcal{F}$ no longer has any factors containing X, so no factor generated afterwards will contain X in its domain. Now, consider a cluster $\mathbf{C}$ downstream from $\mathbf{C}_X$ that contains X. We know that X must be in the domain of the factor in $\mathbf{C}$. We also know that X is not eliminated in $\mathbf{C}$. Therefore, the upstream message from $\mathbf{C}$ must have X in its domain. By definition, the next cluster upstream multiplies in the message from $\mathbf{C}$ (that's how we defined the edges in the cluster tree). Hence, it will also have X in its domain. The same argument holds until $\mathbf{C}_X$ is reached. ∎

Corollary 1.6: *Let $\mathcal{T}$ be a cluster tree induced by a variable elimination algorithm over some Bayesian network G. The separator on an edge in a cluster is precisely the intersection between its two neighboring clusters.*

2 Clique trees

So far, we have used the variable elimination algorithm as a starting point. The algorithm was associated with certain data structures and communication (message passing) structures. These, in turn, induced a cluster tree. We now discuss a somewhat different approach, where we our starting point is a cluster tree. We then use the cluster tree to do variable elimination using the data and communication structures that it defines. As we will see, the same predefined cluster tree can be used in many different ways.

More specifically, we showed about that variable elimination ordering induces a cluster tree for the network that satisfied several properties. It turns out that the converse also holds, any cluster tree that satisfies these properties can be used for determining a variable elimination ordering. In fact, we can use it do variable elimination in a variety of different orders.

Definition 2.1: Let G be a Bayesian network structure. A cluster tree $\mathcal{T}$ is a *clique tree (of G)* if it satisfies the following properties:

- $\mathcal{T}$ is a tree of clusters, where each cluster is a subset of variables in G.
- The separator on an edge C_i—C_j is the intersection $C_i \cap C_j$.
- $\mathcal{T}$ is faithful to G.
- $\mathcal{T}$ satisfies the running intersection property.

∎

We can understand the use of the word "clique" in two ways. Most obviously, in the previous chapter, we said that each factor corresponds to a clique in the induced graph (is either a clique or a subset of it). Thus, every cluster in the cluster tree arising from the variable elimination algorithm corresponds to clique. However, the connection is even deeper. We will see later on that we typically generate a clique tree — one that has the desired properties — by generating an undirected graph over the Bayesian network nodes and constructing a cluster tree whose clusters correspond exactly to (maximal) cliques in this graph.

We start with the most important property of cluster trees:

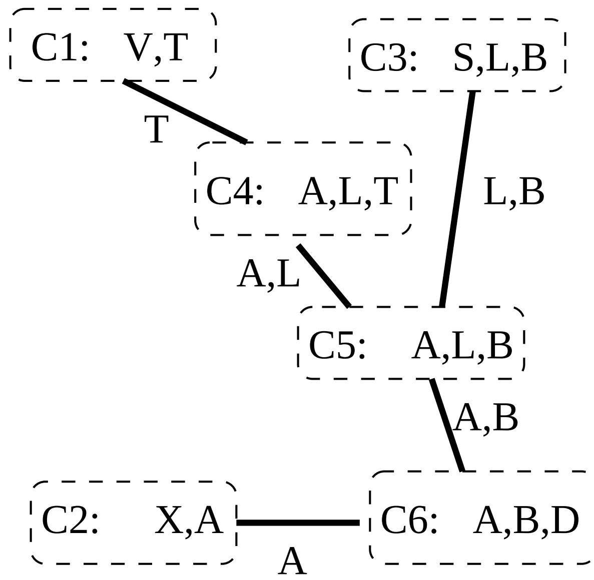

Figure 2: Clique tree for the Asia network.

Theorem 2.2: *Let $\mathcal{T}$ be a clique tree for a network G, and let $S_{i,j}$ be a separator in the graph. Then, $S_{i,j}$ d-separates the graph into two conditionally independent pieces.*

The proof is left as an exercise.

Consider a slightly simplified clique tree $\mathcal{T}$ for the Asia network, shown in Figure 2. Note that it satisfies the two required properties. Assume we want to compute the probability of L. We can do the elimination in an order that's consistent with our data structures in $\mathcal{T}$. For example:

- We eliminate X in $\mathbf{C}_2$ by summing out $P(X \mid A)$, and send a message $\mu_{2\to6}(A)$ $\mathbf{C}_2$ to $\mathbf{C}_6$.
- We eliminate D in $\mathbf{C}_6$ by multiplying $\mu_{2\to6}(A)$ and $P(D \mid A, B)$, and send a message $\mu_{6\to5}(A, B)$ to $\mathbf{C}_5$.
- We eliminate S in $\mathbf{C}_3$ by multiplying $P(S)$, $P(B \mid S)$ and $P(L \mid S)$, and send a message $\mu_{3\to5}(L, B)$ to $\mathbf{C}_5$.
- We eliminate T in $\mathbf{C}_1$ by summing out $P(T \mid V)$ and send a message $\mu_{1\to4}(T)$ to $\mathbf{C}_4$.
- We eliminate A in $\mathbf{C}_4$ by multiplying $\mu_{1\to4}(T)$ and $P(A \mid L, T)$, and send a message $\mu_{4\to5}(A, L)$ to $\mathbf{C}_5$.

At this point, $\mathbf{C}_5$ has received three messages $\mu_{6\to5}(A, B)$, $\mu_{3\to5}(L, B)$, and $\mu_{4\to5}(A, L)$. Looking at this algorithm from the variable elimination perspective, These are the only three remaining factors. Hence, if we multiply them, we get a factor which is the joint probability over A, L, B. To get the marginal over L, we simply eliminate A and B from this factor.

There are several aspects to note about this algorithm.

- We chose to extract $P(L)$ in $\mathbf{C}_5$; $\mathbf{C}_5$ is called the *root* of this computation. All messages go *upstream* — towards the root.
- We could have done the elimination in a variety of orderings. The only constraint is that a clique get all of its downstream messages before it sends its upstream message. We call such cliques *ready*.

- The messages that go along an edge are always factors over the separator.
- We could have chosen any clique that contains L as the root in order to get $P(L)$.
- The same clique tree can be used for computing the probability of any other variable. We simply pick a clique where the variable appears, and eliminate towards that clique.

These points give rise to the more general algorithm. We assume that $\mathcal{T}$ is a clique tree. We begin by assigning each CPD to a clique that contains all the family variables. (We know that such a clique exists because of faithfulness property.) Formally, suppose we use the function $\phi : \{1, \ldots, n\} \mapsto \{1, \ldots, k\}$ to be mapping from CPDs to cliques. Then we can define *clique potentials*

$$\pi_0[C_j] = \prod_{i:\phi(i)=j} P(X_i \mid \mathbf{Pa_i})$$

Since each CPD is assigned to exactly one clique, we have that

$$P(X_1, \ldots, X_n) = \prod_i P(X_i \mid \mathbf{Pa_i}) = \prod_{\mathbf{j}} \pi_{\mathbf{0}}[\mathbf{C_j}]$$

(Note, that this is equivalent to saying that we are representing the joint distribution by a Markov network with potentials $C_1, \ldots, C_k$.)

Now we perform variable elimination with these factors starting from the leaves of the clique tree and moving inward. More precisely, given a query variable Q, we pick some clique containing Q to be the root clique. All cliques send messages directed towards the clique. A clique $\mathbf{C}$ sends a message $\mu_{\mathbf{C}\to\mathbf{C}^+}()$ to its upstream neighbor $\mathbf{C}^+$ via the following computation: it multiplies all incoming messages with its own assigned CPDs, and then sums out all variables except those in the separator between $\mathbf{C}$ and $\mathbf{C}^+$.

We can easily extend this algorithm to accommodate evidence. We use exactly the same approach as we did in variable elimination. We simply reduce all CPDs to make the compatible with the evidence. (Equivalently, we reduce the clique potentials to be compatible with the evidence.) It is easy to see that this approach is correct, for the same reason that it was correct for the case of variable elimination.

The formal version of the algorithm is shown in Figure 3. As we can see, the algorithm maintains a data structure $\pi[\mathbf{C}]$ for each clique $\mathbf{C}$. This data structure is called a *(clique) potential*. It initially contains the product of the CPDs assigned to $\mathbf{C}$. When $\mathbf{C}$ gets all of the messages from its downstream neighbors, it multiplies them into $\pi[\mathbf{C}]$, and sends the appropriate message to its upstream clique. When the root clique $\mathbf{C}_r$ has all messages, it multiplies them into $\pi[\mathbf{C}_r]$; as it has no upstream neighbors, the algorithm terminates. The probability of the query variable Q can then be extracted from $\mathbf{C}_r$ by summing out.

3 Calibration

We have shown that we can use the same clique tree to compute the probability of any node in the graph. In many real-world situations, we often want the probability of a large number of variables. For example, in a medical diagnosis setting, we often want the probability of a large number of possible diseases. When doing speech recognition, we want the probability of all of the phonemes in the word we are trying to recognize.

Assume we want to compute the posterior probability of every random variable in the network. Them most naive approach is to do inference separately for each variable. An approach which is

Procedure Clique-tree-up(
  Graph over $X_1, \ldots, X_n$, // Bayesian network structure
  $P(X_i \mid \mathbf{Pa}_{\mathbf{X_i}})$, // CPDs for Bayesian network nodes
  $u_1, \ldots, u_m$, // evidence
  Q, // query variable
  $\mathcal{T}$ // clique tree for G)
  For each clique $\mathbf{C}$
    Initialize $\pi_0[\mathbf{C}]$ to be the all 1 factor
  For each variable X
    Let $\mathbf{C}$ be some clique that contains $\mathbf{Family_X}$
    $\pi_0[\mathbf{C}] := \pi_0[\mathbf{C}] \cdot P(X \mid \mathbf{Pa_X})|_{\mathbf{U=u}}$
  Let $\mathbf{C}_r$ be some clique that contains Q
  Repeat
    Let $\mathbf{C}$ be a ready clique
    Let $\mathbf{C}_1, \ldots, \mathbf{C}_k$ be $\mathbf{C}$'s downstream neighbors
    Let $\mathbf{C}^+$ be $\mathbf{C}$'s upstream neighbor
    $\pi[\mathbf{C}] := \pi_0[\mathbf{C}] \cdot \prod_{j=1}^{k} \mu_{\mathbf{C}_i \to \mathbf{C}}()$
    Let $\mathbf{Y} = \mathbf{C} \cap \mathbf{C}'$
    Let $\mu_{\mathbf{C} \to \mathbf{C}^+}(\mathbf{Y}) = \sum_{\mathbf{C}-\mathbf{Y}} \pi[\mathbf{C}]$
  Until $\mathbf{C}_r$ has been processed
  Return $\sum_{\mathbf{C}_r - \{Q\}} \pi[\mathbf{C}_r]$

Figure 3: Clique tree elimination

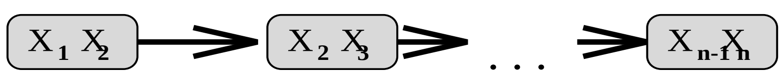

Figure 4: Clique tree for a chain-structured Bayesian network

slightly less naive is to run the algorithm once for every clique, making it the root. However, it turns out that we can do substantially better than either of these.

To understand the idea, let's go back to the case of inference on a chain. Recall that the variable elimination algorithm there involved a computation

$$P(X_{k+1}) = P(X_{k+1} \mid X_k)P(X_k)$$

The associated clique tree has the form shown in Figure 4.

As we discussed, we can make any clique in this tree the root, and sum out the other cliques towards that. Let's assume that we want to compute the probability of X_4. We make $\mathbf{C}_3$ the root, and do the appropriate computation. The message $\mu_{1\to 2}(X_2)$ is computed by multiplying $P(X_1)$ and $P(X_2 \mid X_1)$ and summing out X_1. The message $\mu_{2\to 3}(X_3)$ is computed by multiplying $\mu_{1\to 2}(X_2)$ with $P(X_3 \mid X_2)$ and summing out X_2. Now, assume we want to compute the probability of X_5. We make $\mathbf{C}_4$ the root, and again pass messages. The message $\mu_{1\to 2}(X_2)$ is computed by multiplying $P(X_1)$ and $P(X_2 \mid X_1)$ and summing out X_1. The message $\mu_{2\to 3}(X_3)$ is computed by multiplying $\mu_{1\to 2}(X_2)$ with $P(X_3 \mid X_2)$ and summing out X_2. In other words, the process is exactly the same! Thus, if we want to compute both $P(X_4)$ and $P(X_5)$, there is no point repeating an identical computation for both. This is precisely another situation where dynamic programming is helpful.

So, how would we get all of the probabilities on a chain? We need to compute the messages on all edges, in both directions. On the chain, this requires only $2(n-1)$ computations, where $n-1$ is the number of cliques in the chain. We simply do one forward propagation, computing all forward messages, that go from the beginning of the chain to its end, and one backward propagation, computing all backward messages. Note, however, that we have to be careful. In the algorithm of Figure 3, we create an updated potential when we pass the upstream message. Thus, when doing the forward pass, we would incorporate the forward message into the potential. However, when we are doing the backward pass, we cannot use the updated potentials: if we were doing the simple single-query propagation towards a clique at the beginning of the chain, we would multiply the backward messages into the original potentials. Intuitively, if we use the updated potentials, we would be multiplying CPDs twice: once on the forward pass and once on the backward pass. Thus, when doing the backward pass, we multiply the backward message $\mu_{i+1\to i}(X_{i+1})$ with $\pi_0[\mathbf{C}_i]$, not $\pi[\mathbf{C}_i]$, and use that for producing $\mu_{i\to i-1}(X_i)$.

To compute the final potential at $\mathbf{C}_i$ — the one we would have obtained had we run the algorithm with this clique at the root — we simply multiply $\pi_0[\mathbf{C}_i]$ with both of the incoming messages.

Let's generalize this algorithm to general clique trees. Consider two neighboring cliques $\mathbf{C}_i$ and $\mathbf{C}_j$. The key insight is that here, just as in a chain, the message sent from $\mathbf{C}_i$ to $\mathbf{C}_j$ does not depend on the root. As long as the root is on the "$\mathbf{C}_j$-side", then $\mathbf{C}_i$ sends it exactly the same message. On the other hand, if the root is on the "$\mathbf{C}_i$-side", then $\mathbf{C}_j$ will send exactly the same message no matter where the root actually is. Thus, each edge has two messages associated with it: one for each direction of travel. If we have a total of c cliques, there are $c-1$ edges in the tree; therefore, we have $2(c-1)$ messages to compute.

We can make sure we compute both messages for each edge by the following simple algorithm. First, recall that a message $\mu_{i\to j}()$ from $\mathbf{C}_i$ to $\mathbf{C}_j$ can be computed as soon as $\mathbf{C}_i$ has received messages from all its neighbors except (perhaps) for $\mathbf{C}_j$. When we used the algorithm in Figure 3, we picked a root, and all messages were sent towards it, with a message being sent as soon as all other incoming messages were ready. Let's do the same thing: pick a root and send all messages towards it. The result of this *upward pass* is shown in Figure 5.

When this process is complete, the root has all messages. Therefore, it can now send the appropriate message to all of its children. In Figure 6, it is sending a message to one of its children, based on the messages from the other children and its initial potential.

As soon as it does that, all of its children have all of the information they need to send the messages to their children, so they do so.

This algorithm continues until the leaves of the tree are reached, at which point no more messages need to be sent. This second phase is called the *downward pass*.

At the end of this process, we can compute the final potential for all cliques in the tree, by multiplying the initial potential with each of the incoming messages. The result at each clique $\mathbf{C}_i$ is the probability $P(\mathbf{C}_i, \mathbf{u})$, where $\mathbf{u}$ is our evidence. We can compute the probability $P(X, \mathbf{u})$ by picking a clique in which X appears, and marginalizing out the other variables. Note that if a variable X appears in both $\mathbf{C}_i$ and $\mathbf{C}_j$, then the result of this process will be the same no matter which clique we choose to use. A clique tree for which this property holds is said to be *calibrated*.

Note that this algorithm allows us to compute the probability of all variables in the Bayesian network using twice the computation of variable elimination — an upward pass and a downward pass.

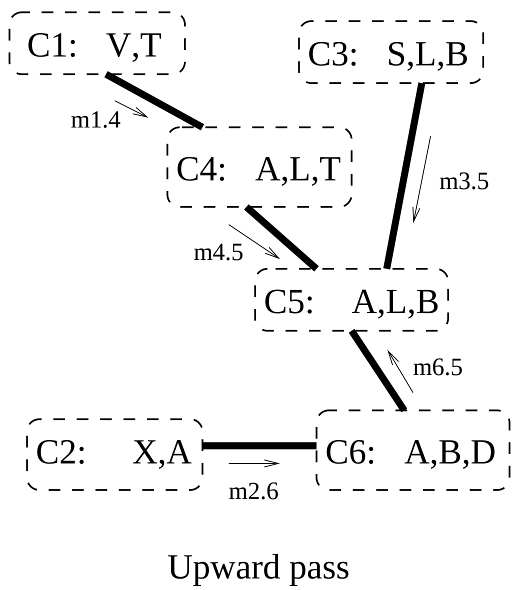

Figure 5: A possible upward pass in the Asia network.

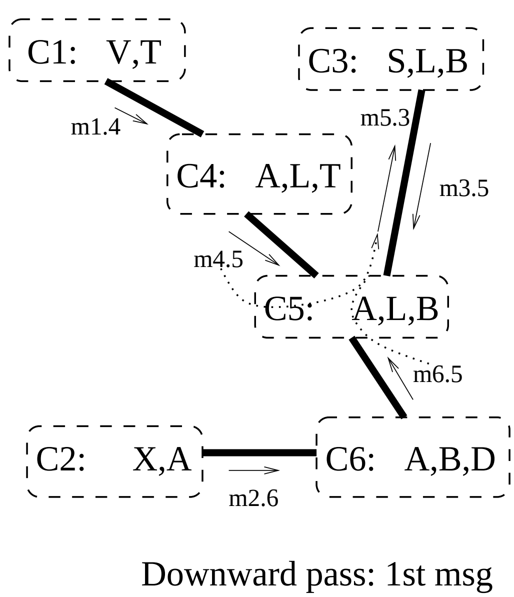

Figure 6: A possible first downward message.

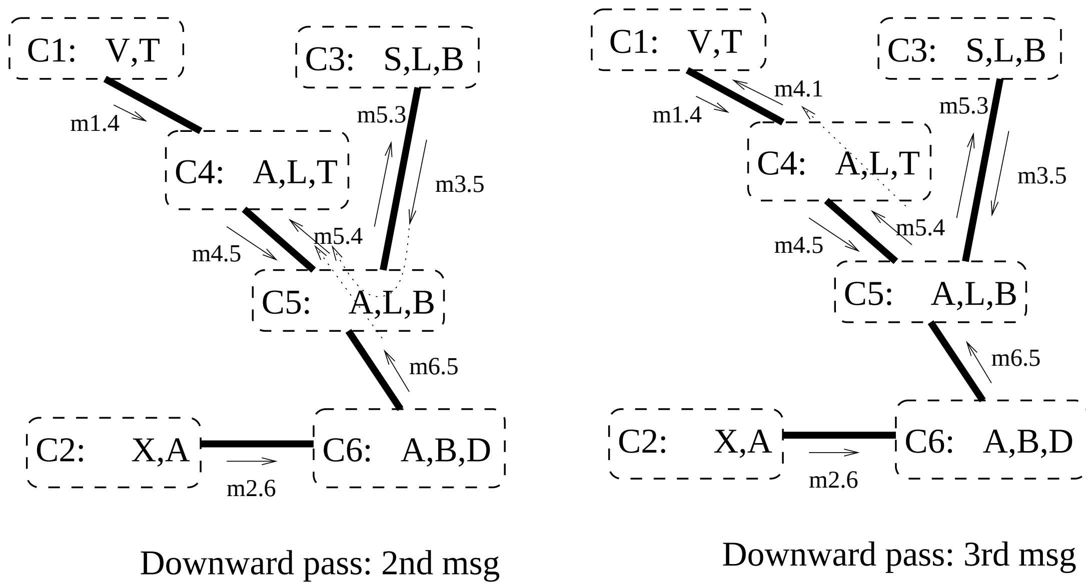

Figure 7: The downward pass continued.

4 Constructing a clique tree

In the previous chapter, we showed that there is a direct correspondence between the maximal factors generated by our algorithm and cliques in the induced graph. In fact, the correspondence is even closer than it first appears. We will show that all induced graphs have a certain property — they are all *chordal*. In the next section, we will show that all chordal graphs can be used to define an elimination ordering where the induced graph is (a subset of) the chordal graph.

Intuitively, an undirected graph is chordal if it contains no cycle of length greater than three that has no "shortcut", i.e., every minimal cycle in the graph is of length three. More precisely:

Definition 4.1: An undirected graph H is *chordal* if for all cycles in H X_1—X_2—…—X_k—X_1 for $k > 3$, there is some edge X_i—X_j besides the edges defining the cycle. ∎

There is a deep connection between induced graphs and chordal graphs. On the one hand, we can show that every induced graph is chordal.

Theorem 4.2: *Every induced graph is chordal.*

Proof: Assume by contradiction that we have such a cycle X_1—X_2—…—X_k—X_1 for $k > 3$, and assume without loss of generality that X_1 is the first variable to be eliminated. As in the proof of Theorem **??** from the previous chapter, both edges X_1—X_2 and X_1—X_k must exist at this point. Therefore, the edge X_2—X_k will be added at the same time, contradicting our assumption. ∎

On the other hand, we can take any chordal graph H that is a superset of the moralized graph, and use it to construct a clique tree. If we do variable elimination on the resulting clique tree, the associated induced graph is exactly H.

The process of taking an undirected graph and finding a chordal superset of it is called *triangulation*. The algorithm is as follows:

1. We take the Bayesian network graph G and moralize it, getting an undirected graph H.

2. We triangulate the graph H to get a chordal graph H'.
3. We find cliques in H', and make each one a node in our clique tree $\mathcal{T}$.
4. We added edges between the cliques in $\mathcal{T}$ to enforce the running intersection property.

We can then use the resulting clique tree for inference, exactly as described above.

There are several steps that we left unspecified in this description.

- **The triangulation step (2).** It turns out that this is the hard step. Finding an optimal triangulation — one that induces small cliques — is NP-hard. This is not surprising, as this is the step that corresponds to finding an optimal elimination ordering in the variable elimination ordering. In fact, the algorithms that find elimination orderings are precisely the same algorithms that find triangulations: We simply generate the induced graph for the ordering; Theorem 4.2 guarantees that it is chordal.
- **Finding maximal cliques (3).** In chordal graphs, this step is easy. One easy approach is to find, for each node, the clique that contains its family. We start with the family, and then add edges until we can't grow the clique any more (i.e., we cannot add any more nodes without violating the fully-connected requirement).
- **Adding edges (4).** We can accomplish this by a maximal spanning tree procedure; intuitively, we connect cliques that have the most variables in common. The procedure takes quadratic time in the number of cliques.

5 Comparison between the algorithms

It is interesting to compare the clique tree and the variable elimination algorithms. In principle, they are equivalent:

- they both use the same basic operations of multiplying factors and summing out variables;
- the algorithms for triangulating the graph are the same as the ones for finding an elimination ordering;
- hence, the overall complexity of the two algorithms is the same.

However, in practice they offer very different advantages and disadvantages. One the one hand:

- The clique tree allows a nontrivial fraction of the operations to be performed in advance, and not during inference for any query: the choice of triangulation/elimination ordering, the product of the CPDs in a single clique.
- The clique tree is designed to allow multi-directional inference using a single upward and downward pass, making multi-query inference more efficient. As we will see, the ability to do multi-query inference is quite important in the context of learning with incomplete data.
- The clique tree data structure can be made *incremental*: when we do inference, the results are stored in the cliques; as new evidence comes in, we do not have to redo all of the inference. It can also be modified to be *lazy*: we only do the computation required for the specific query we have right now.

On the other hand:

- Clique trees are more expensive in terms of space. In a clique tree, we keep all intermediate factors, whereas in variable elimination we can throw them out. If there are c cliques, the cost can be as much as $2c$ times as expensive.
- In a clique tree, the computation structure is fixed and predetermined. We therefore have a lot less flexibility to take advantage of computational efficiencies that arise because of specific features of the evidence and query. For example, in the Asia network, the VE algorithm avoided introducing the dependence between B and L, resulting in substantially less computation. In the clique tree algorithm, the clique structure was predetermined, and the message between $\mathbf{C}_3$ and $\mathbf{C}_5$ remains a factor over B and L. This difference can be quite dramatic in situations where there is a lot of evidence.
- As we will discuss in the next chapter, this type of situation-specific simplification occurs even more often in networks that exhibit context-specific independence. It is even harder to design clique trees that can deal with that case.
- As discussed, clique trees are almost always designed with the cliques as the maximal cliques in a triangulated graph. This sometimes lead to multiplying unnecessarily large factors. For example, by folding in $\mathbf{C}_7$ into $\mathbf{C}_6$ in the Asia network, we caused the message from $\mathbf{C}_2$ to be multiplied with a factor over the three variables A, D, B rather than the factor A, D, hence using more products.