3.1 Review of Last Lecture

3.1.1 Bayes Ball Algorithm

In the last lecture, we concluded with a discussion of the Bayes Ball algorithm, which specifies a set of rules to determine the conditional independence properties of a directed graphical model (DGM). Briefly, to determine if \( X_A \perp \perp X_B \mid X_C \) is true we follow the steps enumerated below,

1. Shade all nodes of \( X_C \) in the graph.
2. Place a ball somewhere in \( X_A \) and allow the ball to bounce on the graph according to certain rules that depend on if nodes are shaded, and on arrow direction (these are discussed below).
3. If there is some staring point where it is possible for the ball to reach \( B \) starting in \( A \), then assert \( X_A \perp \perp X_B \mid X_C \). Otherwise assert, \( X_A \perp \perp X_B \mid X_C \).

The following are the Bayes ball rules. It is easily noticed that they are similar to the definition of d-separation. These are rules to determine what can happen when a ball from node \( X \) arrives at node \( Y \) on its (attempted) way to a node \( Z \). All other nodes that might be joined to \( Y \) are ignored for now. Note that it might be possible for a ball to bounce through \( Y \) to another node even if it can’t bounce through to \( Z \), and vice versa.

**Bayes Ball Bouncing Rules**

**Case I: Serial arrows at \( Y \) which corresponds to \( X \perp \perp Z \mid Y \)**

In this case, a ball bounces through only if \( Y \) is unshaded.

![Ball goes through](Figure 3.1: Case I i.)

![Ball is blocked](Figure 3.2: Case I ii. \( X \perp \perp Z \mid Y \))
Case II: Diverging arrows at $Y$, which corresponds to $X \perp \perp Z | Y$ again

In this case again, the ball bounces through only if $Y$ is unshaded.

![Figure 3.3: Case II i.](image)

![Figure 3.4: Case II ii. $X \perp \perp Z | Y$](image)

Case III.: converging arrows at $Y$, which corresponds to $X \perp \perp Z$

In this case, the ball bounces through only if $Y$ is shaded. Note that this is different than the other cases.

![Figure 3.5: Case III i. $X \perp \perp Y$](image)

![Figure 3.6: Case III ii. $X \perp \perp Z | Y$](image)

Boundary Conditions

We’ll also need a boundary condition. In this case, the ball bounces back if only the node $Y$ is shaded. Boundary conditions imply that when the ball arrives at leaf nodes, which are not shaded, it dies. This ensures that valid conditional independence statements can be made on general graphs.

![Figure 3.7: Boundary i.](image)

![Figure 3.8: Boundary ii.](image)

3.2 Examples of Bayes Ball Algorithm

Following are some examples of conditional independence properties derived using the Bayes Ball Algorithm for DGMs. For each of the examples, the nodes to the right of the conditioning bar are shaded and then the Bayes Ball rules are applied.
\{X_2, X_1\} \perp \perp \{X_4, X_5\}|X_3
\[X_1 \perp \perp X_5|X_3\]

\[X_4 \not\perp \not\perp \{X_1, X_3\}|X_2\]
\[X_4 \not\perp \{X_1, X_3\}\]
\[X_4 \perp \{X_1, X_3, X_5, X_6\}|X_2\]

\[X_1 \perp \perp X_6|\{X_2, X_3\}\]
This example shows that there might be long term dependencies that aren’t obvious at first, especially if we use the naive notion of graph separation. For this DGM it can be concluded that $X \perp Z | Y$. The joint distribution of all the nodes in this graph can be written as,

$$P(X, Y, Z, A, B, C, D, E) = P(Y|D)P(D|C, E)P(E|Z)P(C|B)P(B|A)P(X|A)$$

which cannot be marginalized to

$$P(X, Y|Z) = P(X|Z)P(Y|Z)$$

However, for this graph $X \perp Z$. 
3.3 View of Family of Distributions

Here are two views of directed graphical models in terms of the set of underlying probability distributions that correspond to them. The basic idea is again that a graphical model represents a family of probability distributions, those distributions that do not violate the conditional independence properties specified by the graph. Here are two notions of family (or sets of probability distributions) that are actually the same.

Define

\[ F_1 = \left\{ p(x_1:n) : p(x_1:n) = \prod p(x_i|x_{\pi_i}) \right\} \]

for some given \( p(x_i|x_{\pi_i}) \) and where \( \pi_i \) are the indices of the parents of node \( x_i \) according to the graph. In this case, we exhaustively specify all probability distributions that can be factored according to the graph.

We also define:

\[ F_2 = \left\{ p(x_1:n) : p(x_1:n) \text{ satisfies } I_j \ \forall j \right\} \]

where \( I_j \) is a conditional independence statement that is made by the same graph (i.e., think of running the Bayes ball algorithm a bunch of times until we’ve exhaustively mentioned all possible independence statements, and let the \( j \)th statement be referred to as \( I_j \{ X_{A_j} \perp \perp X_{B_j} | X_{C_j} \} \).

**Theorem:** \( F_1 = F_2 \)

We will prove this later in the course. In any event, it is interesting to note that these two different sets of distributions are the same. The moral of the story, again, is that GMs correspond to families of distributions.

3.4 Inference

The “inference” problem in GMs is calculating the probability distribution of one set of nodes given another. Say that we are given a set of nodes, \( U = \{1 : n\} = E \cup F \cup H \) where \( E, F, H \) are mutually disjoint subsets formed by making arbitrary partitions.

Our goal is to compute

\[ p(X_F|X_E) = p(X_F, X_E)/p(X_E) \]

where

\[ p(X_F, X_E) = \sum_{X_H} p(X_F, X_E, X_H) \]

and

\[ p(X_E) = \sum_{X_F} p(X_F, X_E) \]

where

- \( E \) = evidence nodes, these are the nodes given
- \( F \) = query nodes, the ones we want the dist of
- \( H \) = hidden nodes, the ones we must integrate away

Alternatively, in pattern recognition applications, probabilistic inference can also be expressed as,

\[ X_F^* = \arg\max_{X_F} p(X_F|X_E) \]
since the goal is to determine the most probable query node of the graph given the evidence nodes.

Note that any subset of nodes can be, at different times, either input, hidden, or output (this is unlike a neural network which has fixed rules for each of the nodes). i.e., it is as easy to run a GM backwards as forwards during inference using a message passing protocol.

A special case arises when we have a graph with two nodes only as shown in Fig. 14. For this case the problem of inference is easy. For example, it is easy to calculate \( p(y|x) \) just by a table lookup since due to graph factorization,

\[
p(x, y) = p(x)p(y|x)
\]

where \( p(x, y) \) can be represented by a table. It also isn’t hard to calculate \( p(x|y) \) by Bayes rule, i.e.,

\[
p(x|y) = \frac{p(y|x)p(x)}{p(y)}.
\]

For larger graphs inference is hard. Suppose there are \( n \) nodes, the set \( H = \{6\} \), and all nodes are discrete taking on \( r \) possible values. Then

\[
p(X_F, X_E) = \sum_{X_H} p(X_F, X_E, X_H)
\]

will require \( r^n \) calculations. Ex. even if \( n = 1000 \), and \( r = 3 \) this is prohibitive.

What we can do is exploit the local structure of a problem as it is represented by a DGM. i.e., this example illustrates how to use independence properties to reduce the needed calculation to do inference.

Example:
First, we’ll reduce computation using algebra. Reading conditional independence properties from the above graph, we have:

\[ p(x_{1:6}) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_{2,5}) \]

then

\[ P(x_{1:5}) = \sum_{x_6} p(x_{1:6}) \]

\[ = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3) \sum_{x_6} p(x_6|x_{2,5}) \]

\[ = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3) \sum_{x_6} p(x_6|x_{2,5}) \]

So, this requires only \( r^3 \) calculations i.e.,

\[ P(X_{1:5}) = \sum_{x_6} p(X_{1:6}) \quad \leftarrow \quad O(r^6) \text{ calculations} \]

\[ = p(x_1)p(x_2|x_1) \ldots \sum_{x_5} p(x_6|x_{2,5}) \quad \leftarrow \quad O(r^3) \]

Another thing we might try is:

\[ P(X_{\{1,6\} \setminus \{2\}}) = p(x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3) \sum_{x_6} p(x_6|x_{2,3})p(x_6|x_1) \]

also requiring \( r^3 \) calculations. Eventually, we’ll be able to do inference with graphs instead of algebra.

Here we reduce calculation by exploiting substructure. Say we want to calculate:

\[ P(x_1|x_6) = \frac{p(x_1, x_6)}{p(x_6)} \]

We will use the notation:

\[ \phi_{x_i}(X_A) = \sum_{x_i} f(x_i, X_A) \]

which is only a function of \( X_A \) since \( x_i \) has been marginalized away. Note \( f() \) is some function and where \( x_i \) is given. We can use this in the following (which as we will see will eventually correspond to graph operations).
Elimination is the graphical way of doing the algebraic marginalizations discussed in the previous section. The algorithm works on undirected graphical models (which we haven’t defined yet); suffice it to say for now, directed models must first be “moralized” before performing these steps (we will ultimately treat this much more formally).

Here is the elimination procedure:

1. Choose an ordering on the nodes (elimination ordering).

Note that in each case, summing over a variable has essentially “eliminated” the variable from further consideration. Also notice that since none of the marginalization operation need more than $O(r^3)$ calculations, we have reduced computation from $O(r^6)$ to $O(r^3)$ calculation.

### 3.4.1 Graphical Elimination Algorithm

Elimination is the graphical way of doing the algebraic marginalizations discussed in the previous section. The algorithm works on undirected graphical models (which we haven’t defined yet); suffice it to say for now, directed models must first be “moralized” before performing these steps (we will ultimately treat this much more formally).

Here is the elimination procedure:

1. Choose an ordering on the nodes (elimination ordering).
2. Choose next node in elimination order, connect all neighbors of the node (all pairs of neighbors are connected).

3. Eliminate that node and its edges from the graph.

![Elimination Example](image)

Figure 3.16: Elimination Example. \( I = \{6, 5, 4, 3, 2, 1\} \)

Note that the reconstructed graph after elimination but with all the edges that have been added is called “triangulated” or “chordal” and has many important theoretical properties that, again, we will use later in the term.
We define elimination cliques as:

\[ C_i = X_i \bigcup \{ \text{set of all nodes that are neighbors of } X_i \text{ at elimination time} \} \]

Fig 3.16 shows an example of the elimination algorithm on an undirected graph with the elimination order \( \{6, 5, 4, 3, 2, 1\} \). In going from Fig 3.16(a) to Fig 3.16(b), \( X_2 \) and \( X_5 \), which are neighbors of \( X_6 \), are connected and \( X_6 \) is subsequently eliminated in Fig 3.16(c). Similarly, the neighbors of \( X_5 \) are connected in Fig 3.16(c) and \( X_5 \) is eliminated in Fig 3.16(d). Next \( X_4 \) is eliminated without any need to connect its neighbors. Since all the nodes in the eliminated graph at this stage are connected with each other \( X_3 \) and \( X_2 \) are eliminated in Fig 3.16(f) and Fig 3.16(g) without needing to connect any neighbors. In the example above, the elimination cliques are:

\[
\begin{align*}
C_6 &= \{x_2, x_5, x_6\} \\
C_5 &= \{x_2, x_3, x_5\} \\
C_4 &= \{x_4, x_2\} \\
C_3 &= \{x_3, x_2, x_1\} \\
C_2 &= \{x_2, x_1\} \\
C_1 &= \{x_1\}
\end{align*}
\]

**Key Point:** The key point is that size of the elimination clique determines the number of operations. For example,

\[
\phi_{x_3}(x_2, x_1) = \sum_{x_3} p(x_3|x_1) \phi_{x_5}(x_2, x_3) \Rightarrow O \left( r^3 \right)
\]

and \( C_3 = \{1, 2, 3\} \) i.e. the number of operations is \( r^{|C|} \).

For optimality, we would like to choose the ordering that minimizes the size of the cliques, but this is an NP-Hard problem, so we must in general come up with elimination order heuristics.

But we need an elimination algorithm on DGMs for now.

- Directed graphs express probabilities conditioned on parents, \( \pi_i \).
- Directed graph parents are not necessarily connected (\( x_6 \) in previous example whose parents are not connected). This typically happens when there is a v-structure in a DGM. Thus, elimination performed directly on a directed graph might make assumptions that were not correct according to the original DGM. Also the elimination cliques formed would not be the correct determining units of computation.
- Therefore, we must “moralize” the DGM before elimination. Moralization means to connect any unconnected parents of any nodes, and then drop the directions on the arrows.
- Note that this operation is valid because by adding edges we are only making fewer independence statements, and therefore solving the inference problem for a larger family of probability distributions.

**Example:**
3.4.2 Generalized Probabilistic Inference algorithm

What we are going to do is come up with an algorithm that, in the general case, corresponds both to a graph elimination algorithm and to the actual equations needed to compute a probabilistic inference. While this example and algorithm is meant only to give you a flavor of inference (there are a bunch of details that will be left out until later in the quarter), you will still see the basic flavor of the algorithm. Note that this procedure will make use of the elimination cliques generated from the elimination algorithm run on the moralized DGM.

First, a definition:

**Definition:** Each node has a potential function, \( \Psi_{x_i}(x_{C_i}) \), associated with the elimination clique, \( C_i \). This can be viewed as a \(|C_i|\)-dimensional table of values.

Like before we will sum over \( x_i \) to produce \( \phi \) functions. Recall that

\[
\phi_{X_i}(A) = \sum_{x_i} f(x_i, A)
\]

so the key operation that we will perform will be:

\[
\phi_{X_i}(X_{C_i'}) = \sum_{x_i} \psi_{x_i}(x_{C_i})
\]

where \( C_i' \equiv C_i \setminus \{i\} \) which is the elimination clique without \( i \).

**Inference Algorithm:** (see handout 2) We input \( E \), and a single query node \( F \).

1. Initialize: \( \psi_{x_i}(x_{C_i}) = p(x_i|x_{\pi_i}) \) for each clique.
2. Find a good elimination order, where the query node \( F \) is last in that order.
3. choose the next node in the ordering, say \( X_i \).
4. If \( i \notin E \) (so not an evidence node) then construct \( \psi_{X_i}(x_{C_i}) \) (by multiplying factors if need be) and then compute:

\[
\phi_{X_i}(X_{C_i'}) = \sum_{x_i} \psi_{x_i}(X_{C_i})
\]
5. If, on the other hand, \( i \in E \), then construct \( \phi_{X_i}(x_{C_i}) \) as a product of components of \( \psi_{X_i}(x_{C_i}) \) containing \( X_i = x_i \).

6. Find node \( X_k \) in \( X_{C_i} \) whose index appears first (i.e., earliest in list order) in the eliminating order. Assign \( \phi_{X_i}(x_{C_i'}) \) to the potential function at \( x_k \).

The main thing to realize is that the set of equations that we derived for inference correspond to a set of graph operations, since each marginalizing operation over a node is equivalent to eliminating that node. Ultimately, what we will want to do is to come up with a set of purely graph operations and inference procedures which are relatively easy to do and which correspond to valid (and efficient) manipulation of the set of equations.

References