CS 188 Introduction to Artificial Intelligence
Spring 2022 Note 5

These lecture notes are based on notes originally written by Josh Hug and Jacky Liang. They have been heavily updated by Regina Wang.

Probability Rundown

We're assuming that you've learned the foundations of probability in CS70, so these notes will assume a basic understanding of standard concepts in probability like PDFs, conditional probabilities, independence, and conditional independence. Here we provide a brief summary of probability rules we will be using.

A **random variable** represents an event whose outcome is unknown. A **probability distribution** is an assignment of weights to outcomes. Probability distributions must satisfy the following conditions:

$$0 \leq P(\omega) \leq 1$$
$$\sum_{\omega} P(\omega) = 1$$

For instance if A is a binary variable (can only take on two values) then $P(A = 0) = p$ and $P(A = 1) = 1 - p$ for some $p \in [0, 1]$.

We will use the convention that capital letters refer to random variables and lowercase letters refer to some specific outcome of that random variable.

We use the notation $P(A, B, C)$ to denote the **joint distribution** of the variables A, B, C. In joint distributions ordering does not matter i.e. $P(A, B, C) = P(C, B, A)$.

We can expand a joint distribution using the **chain rule**, also sometimes referred to as the product rule.

$$P(A, B) = P(A|B)P(B) = P(B|A)P(A)$$
$$P(A_1, A_2 ... A_k) = P(A_1)P(A_2|A_1)...P(A_k|A_1...A_{k-1})$$

The **marginal distribution** of A, B can be obtained by summing out all possible values that variable C can take as $P(A, B) = \sum_c P(A, B, C = c)$. The marginal distribution of A can also be obtained as $P(A) = \sum_b \sum_c P(A, B = b, C = c)$. We will also sometimes refer to the process of marginalization as "summing out".

When we do operations on probability distributions, sometimes we get distributions that do not necessarily sum to 1. To fix this, we **normalize**: take the sum of all entries in the distribution and divide each entry by that sum.

Conditional probabilities assign probabilities to events conditioned on some known facts. For instance $P(A|B = b)$ gives the probability distribution of A given that we know the value of B equals b. Conditional probabilities are defined as:

$$P(A|B) = \frac{P(A, B)}{P(B)}.$$

Combining the above definition of conditional probability and the chain rule, we get the **Bayes Rule**:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

To write that random variables A and B are **mutually independent**, we write $A \perp\!\!\!\perp B$. This is equivalent to $B \perp\!\!\!\perp A$.

When A and B are mutually independent, $P(A,B) = P(A)P(B)$. An example you can think of are two independent coin flips. You may be familiar with mutual independence as just 'independence' in other courses. We can derive from the above equation and the chain rule that $P(A|B) = P(A)$ and $P(B|A) = P(B)$.

To write that random variables A and B are **conditionally independent** given another random variable C, we write $A \perp\!\!\!\perp B|C$. This is also equivalent to $B \perp\!\!\!\perp A|C$.

If A and B are conditionally independent given C, then $P(A,B|C) = P(A|C)P(B|C)$. This means that if we have knowledge about the value of C, then B and A do not affect each other. Equivalent to the above definition of conditional independence are the relations $P(A|B,C) = P(A|C)$ and $P(B|A,C) = P(B|C)$. Notice how these three equations are equivalent to the three equations for mutual independence, just with an added conditional on C!

Probabilistic Inference

In artificial intelligence, we often want to model the relationships between various nondeterministic events. If the weather predicts a 40% chance of rain, should I carry my umbrella? How many scoops of ice cream should I get if the more scoops I get, the more likely I am to drop it all? If there was an accident 15 minutes ago on the freeway on my route to Oracle Arena to watch the Warriors' game, should I leave now or in 30 minutes? All of these questions (and many more) can be answered with **probabilistic inference**.

In previous sections of this class, we modeled the world as existing in a specific state that is always known. For the next several weeks, we will instead use a new model where each possible state for the world has its own probability. For example, we might build a weather model, where the state consists of the season, temperature and weather. Our model might say that $P(winter, 35^\circ, cloudy) = 0.023$. This number represents the probability of the specific outcome that it is winter, 35°, and cloudy.

More precisely, our model is a **joint distribution**, i.e. a table of probabilities which captures the likelihood of each possible **outcome**, also known as an **assignment** of variables. As an example, consider the table below:

Season	Temperature	Weather	Probability
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

This model allows us to answer questions that might be of interest to us, for example:

- What is the probability that it is sunny? $P(W = sun)$
- What is the probability distribution for the weather, given that we know it is winter? $P(W|S = winter)$
- What is the probability that it is winter, given that we know it is rainy and cold? $P(S = winter|T = cold, W = rain)$
- What is the probability distribution for the weather and season give that we know that it is cold? $P(S, W|T = cold)$

Inference By Enumeration

Given a joint PDF, we can trivially compute any desired probability distribution $P(Q_1 \dots Q_k|e_1 \dots e_k)$ using a simple and intuitive procedure known as **inference by enumeration**, for which we define three types of variables we will be dealing with:

1. **Query variables** Q_i, which are unknown and appear on the left side of the conditional (|) in the desired probability distribution.
2. **Evidence variables** e_i, which are observed variables whose values are known and appear on the right side of the conditional (|) in the desired probability distribution.
3. **Hidden variables**, which are values present in the overall joint distribution but not in the desired distribution.

In Inference By Enumeration, we follow the following algorithm:

1. Collect all the rows consistent with the observed evidence variables.
2. Sum out (marginalize) all the hidden variables.
3. Normalize the table so that it is a probability distribution (i.e. values sum to 1)

For example, if we wanted to compute $P(W|S = winter)$ using the above joint distribution, we'd select the four rows where S is winter, then sum out over T and normalize. This yields the following probability table:

W	S	Unnormalized Sum	Probability
sun	winter	$0.10 + 0.15 = 0.25$	$0.25/(0.25 + 0.25) = 0.5$
rain	winter	$0.05 + 0.20 = 0.25$	$0.25/(0.25 + 0.25) = 0.5$

Hence $P(W = sun|S = winter) = 0.5$ and $P(W = rain|S = winter) = 0.5$, and we learn that in winter there's a 50% chance of sun and a 50% chance of rain.

So long as we have the joint PDF table, inference by enumeration (IBE) can be used to compute any desired probability distribution, even for multiple query variables $Q_1 ... Q_k$.

Bayesian Network Representation

While inference by enumeration can compute probabilities for any query we might desire, representing an entire joint distribution in the memory of a computer is impractical for real problems — if each of n variables we wish to represent can take on d possible values (it has a **domain** of size d), then our joint distribution table will have d^n entries, exponential in the number of variables and quite impractical to store!

Bayes nets avoid this issue by taking advantage of the idea of conditional probability. Rather than storing information in a giant table, probabilities are instead distributed across a number of smaller conditional probability tables along with a **directed acyclic graph** (DAG) which captures the relationships between variables. The local probability tables and the DAG together encode enough information to compute any probability distribution that we could have computed given the entire large joint distribution. We will see how this works in the next section

We formally define a Bayes Net as consisting of:

- A directed acyclic graph of nodes, one per variable X.
- A conditional distribution for each node $P(X|A_1 \dots A_n)$, where A_i is the i^{th} parent of X, stored as a **conditional probability table** or CPT. Each CPT has $n+2$ columns: one for the values of each of the n parent variables $A_1 \dots A_n$, one for the values of X, and one for the conditional probability of X given its parents.

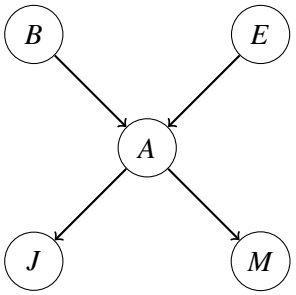
The structure of the Bayes Net graph encodes conditional independence relations between different nodes. These conditional independences allow us to store multiple small tables instead of one large one.

It is important to remember that the edges between Bayes Net nodes do not mean there is specifically a *causal* relationship between those nodes, or that the variables are necessarily dependent on one another. It just means that there may be *some* relationship between the nodes.

As an example of a Bayes Net, consider a model where we have five binary random variables described below:

- B: Burglary occurs.
- A: Alarm goes off.
- E: Earthquake occurs.
- J: John calls.
- M: Mary calls.

Assume the alarm can go off if either a burglary or an earthquake occurs, and that Mary and John will call if they hear the alarm. We can represent these dependencies with the graph shown below.



In this Bayes Net, we would store probability tables $P(B), P(E), P(A|B,E), P(J|A)$ and $P(M|A)$.

Given all of the CPTs for a graph, we can calculate the probability of a given assignment using the following rule:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | parents(X_i))$$

For the alarm model above, we can actually calculate the probability of a joint probability as follows: $P(-b,-e,+a,+j,-m) = P(-b) \cdot P(-e) \cdot P(+a|-b,-e) \cdot P(+j|+a) \cdot P(-m|+a)$.

We will see how this relation holds in the next section.

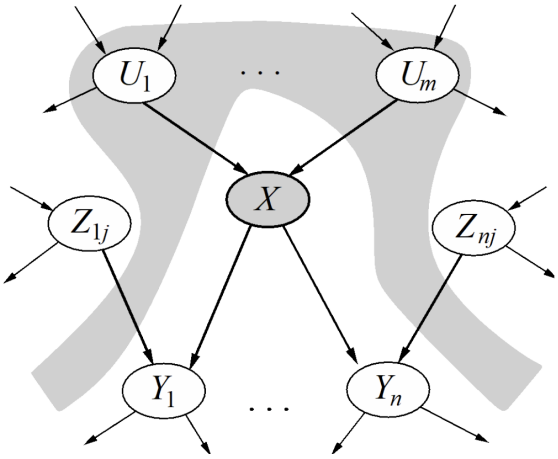
As a reality check, it's important to internalize that Bayes Nets are only a type of model. Models attempt to capture the way the world works, but because they are always a simplification they are always wrong. However, with good modeling choices they can still be good enough approximations that they are useful for solving real problems in the real world.

In general, a good model may not account for every variable or even every interaction between variables. But by making modeling assumptions in the structure of the graph, we can produce incredibly efficient inference techniques that are often more practically useful than simple procedures like inference by enumeration.

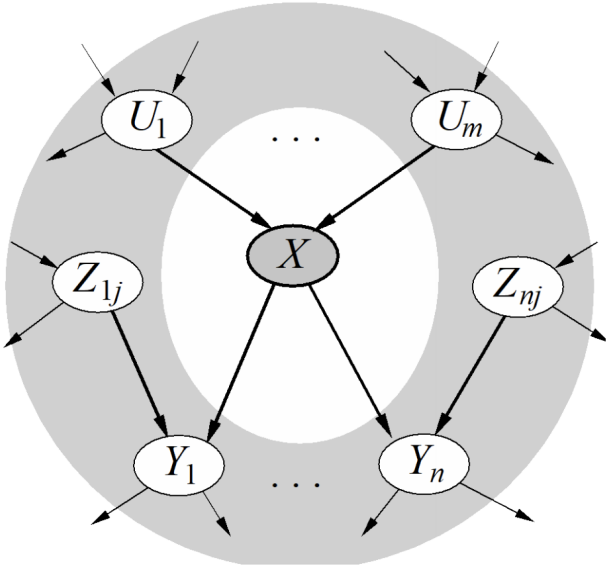
Structure of Bayes Nets

In this class, we will refer to two rules for Bayes Net independences that can be inferred by looking at the graphical structure of the Bayes Net:

- **Each node is conditionally independent of all its ancestor nodes (non-descendants) in the graph, given all of its parents**.



- **Each node is conditionally independent of all other variables given its Markov blanket.** A variable's Markov blanket consists of parents, children, children's other parents.

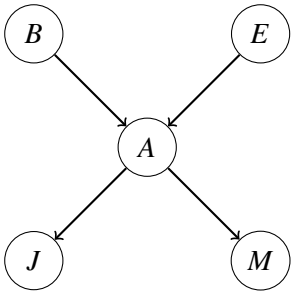


Using these tools, we can return to the assertion in the previous section: that we can get the joint distribution of all variables by joining the CPTs of the Bayes Net.

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | parents(X_i))$$

This relation between the joint distribution and the CPTs of the Bayes net works because of the conditional independence relationships given by the graph. We will prove this using an example.

Let's revisit the previous example. We have the CPTs $P(B), P(E), P(A|B,E), P(J|A)$ and $P(M|A)$, and the following graph:



For this Bayes net, we are trying to prove the following relation:

$$P(B,E,A,J,M) = P(B)P(E)P(A|B,E)P(J|A)P(M|A) \tag{1}$$

We can expand the joint distribution another way: using the chain rule. If we expand the joint distribution with topological ordering (parents before children), we get the following equation

$$P(B,E,A,J,M) = P(B)P(E|B)P(A|B,E)P(J|B,E,A)P(M|B,E,A,J) \tag{2}$$

Notice that in Equation (1) every variable is represented in a CPT $P(var|Parents(var))$, while in Equation (2), every variable is represented in a CPT $P(var|Parents(var), Ancestors(var))$.

We rely on the first conditional independence relation above, that **each node is conditionally independent of all its ancestor nodes in the graph, given all of its parents**[1].

Therefore, in a Bayes net, $P(var|Parents(var), Ancestors(var)) = P(var|Parents(var))$, so Equation (1) and Equation (2) are equal. The conditional independences in a Bayes Net allow for multiple smaller conditional probability tables to represent the entire joint probability distribution.

[1] Elsewhere, the assumption may be defined as "a node is conditionally independent of its *non-descendants* given its parents." We always want to make the minimum assumption possible and prove what we need, so we will use the ancestors assumption.

Exact Inference in Bayes Nets

Inference is the problem of finding the value of some probability distribution $P(Q_1...Q_k|e_1...e_k)$, as detailed in the Probabilistic Inference section at the beginning of the note. Given a Bayes Net, we can solve this problem naively by forming the joint PDF and using Inference by Enumeration. This requires the creation of and iteration over an exponentially large table.

Variable Elimination

An alternate approach is to eliminate hidden variables one by one. To **eliminate** a variable X, we:

1. Join (multiply together) all factors involving X.
2. Sum out X.

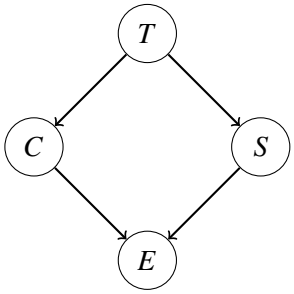
A **factor** is defined simply as an *unnormalized probability*. At all points during variable elimination, each factor will be proportional to the probability it corresponds to but the underlying distribution for each factor won't necessarily sum to 1 as a probability distribution should. The pseudocode for variable elimination is here:

function ELIMINATION-ASK(X, **e**, bn) **returns** a distribution over X
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $factors \leftarrow [\,]$
  for each var **in** ORDER(bn.VARS) **do**
    $factors \leftarrow [\text{MAKE-FACTOR}(var, \mathbf{e})|factors]$
    if var is a hidden variable **then** $factors \leftarrow$ SUM-OUT($var, factors$)
  return NORMALIZE(POINTWISE-PRODUCT($factors$))

Figure 14.11 The variable elimination algorithm for inference in Bayesian networks.

Let's make these ideas more concrete with an example. Suppose we have a model as shown below, where T, C, S, and E can take on binary values, as shown below. Here, T represents the chance that an adventurer takes a treasure, C represents the chance that a cage falls on the adventurer given that he takes the treasure, S represents the chance that snakes are released if an adventurer takes the treasure, and E represents the chance that the adventurer escapes given information about the status of the cage and snakes.



In this case, we have the factors $P(T)$, $P(C|T)$, $P(S|T)$, and $P(E|C,S)$. Suppose we want to calculate $P(T|+e)$. The inference by enumeration approach would be to form the 16 row joint PDF $P(T,C,S,E)$, select only the rows corresponding to +e, then summing out C and S and finally normalizing.

The alternate approach is to eliminate C, then S, one variable at a time. We'd proceed as follows:

- Join (multiply) all the factors involving C, forming $f_1(C,+e|T,S) = P(C|T) \cdot P(+e|C,S)$.
- Sum out C from this new factor, leaving us with a new factor $f_2(+e|T,S)$.
- Join all factors involving S, forming $f_3(+e,S|T) = P(S|T) \cdot f_2(+e|T,S)$.
- Sum out S, yielding $f_4(+e|T)$.
- Join the remaining factors, which gives $f_5(+e,T) = f_4(+e|T) \cdot P(T)$

Once we have $f_5(+e,T)$, we can easily compute $P(T|+e)$ by normalizing.

While this process is more involved from a conceptual point of view, the maximum size of any factor generated is only 8 rows instead of 16 as it would be if we formed the entire joint PDF.

An alternate way of looking at the problem is to observe that the calculation of $P(T|+e)$ can either be done through inference by enumeration as follows:

$$\alpha \sum_s \sum_c P(T)P(s|T)P(c|T)P(+e|c,s)$$

or by Variable elimination as follows:

$$\alpha P(T) \sum_s P(s|T) \sum_c P(c|T)P(+e|c,s)$$

We can see that the equations are equivalent, except that in variable elimination we have moved terms that are irrelevant to the summations outside of each summation!

As a final note on variable elimination, it's important to observe that it only improves on inference by enumeration if we are able to limit the size of the largest factor to a reasonable value.

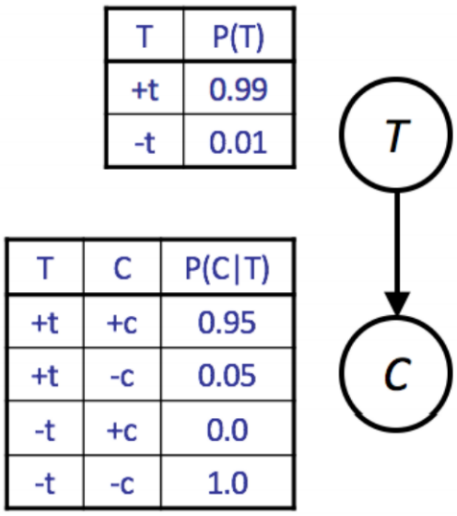
Approximate Inference in Bayes Nets: Sampling

An alternate approach for probabilistic reasoning is to implicitly calculate the probabilities for our query by simply counting samples. This will not yield the exact solution, as in IBE or Variable Elimination, but this approximate inference is often good enough, especially when taking into account massive savings in computation.

For example, suppose we wanted to calculate $P(+t|+e)$. If we had a magic machine that could generate samples from our distribution, we could collect all samples for which $E = +e$, and then compute the fraction of those samples for while $T = +t$. We'd easily be able to compute any inference we'd want just by looking at the samples. Let's see some different methods for generating samples.

Prior Sampling

Given a Bayes Net model, we can easily write a simulator. For example, consider the CPTs given below for the simplified model with only two variables T and C.



A simple simulator in Python would be written as follows:

```
import random

def get_t():
    if random.random() < 0.99:
        return True
    return False

def get_c(t):
    if t and random.random() < 0.95:
        return True
    return False

def get_sample():
    t = get_t()
    c = get_c(t)
    return [t, c]
```

We call this simple approach **prior sampling**. The downside of this approach is that it may require the generation of a very large number of samples in order to perform analysis of unlikely scenarios. If we wanted to compute $P(C|-t)$, we'd have to throw away 99% of our samples.

Rejection Sampling

One way to mitigate the previously stated problem is to modify our procedure to early reject any sample inconsistent with our evidence. For example, for the query $P(C|-t)$, we'd avoid generating a value for C unless t is false. This still means we have to throw away most of our samples, but at least the bad samples we generate take less time to create. We call this approach **rejection sampling**.

These two approaches work for the same reason: any valid sample occurs with the same probability as specified in the joint PDF.

Likelihood Weighting

A more exotic approach is **likelihood weighting**, which ensures that we never generate a bad sample. In this approach, we manually set all variables equal to the evidence in our query. For example, if we wanted to compute $P(C|-t)$, we'd simply declare that t is false. The problem here is that this may yield samples that are inconsistent with the correct distribution.

If we simply force some variables to be equal to the evidence, then our samples occur with probability only equal to the products of the CPTs of the non-evidence variables. This means the joint PDF has no guarantee of being correct (though may be for some cases like our two variable Bayes Net). Instead, if we have sampled variables Z_1 through Z_p and fixed evidence variables E_1 through E_m a sample is given by the probability $P(Z_1...Z_p, E_1...E_m) = \prod_i^p P(Z_i|Parents(Z_i))$. What is missing is that the probability of a sample does not include all the probabilities of $P(E_i|Parents(E_i))$, i.e. not every CPT participates.

Likelihood weighting solves this issue by using a weight for each sample, which is the probability of the evidence variables given the sampled variables. That is, instead of counting all samples equally, we can define a weight w_j for sample j that reflects how likely the observed values for the evidence variables are, given the sampled values. In this way, we ensure that every CPT participates. To do this, we iterate through each variable in the Bayes net, as we do for normal sampling), sampling a value if the variable is not an evidence variable, or changing the weight for the sample if the variable is evidence.

For example, suppose we want to calculate $P(T|+c,+e)$. For the jth sample, we'd perform the following algorithm:

- Set w_j to 1.0, and c = true and e = true.
- For T: This is not an evidence variable, so we sample t_j from $P(T)$.
- For C: This is an evidence variable, so we multiply the weight of the sample by $P(+c|t_j)$, i.e. $w_j = w_j \cdot P(+c|t_j)$.
- For S: sample s_j from $P(S|t_j)$.
- For E: multiply the weight of the sample by $P(+e|+c,s_j)$, i.e. $w_j = w_j \cdot P(+e|+c,s_j)$.

Then when we perform the usual counting process, we weight sample j by w_j instead of 1, where $0 <= w_j <= 1$. This approach works because in the final calculations for the probabilities, the weights effectively serve to replace the missing CPTs. In effect, we ensure that the weighted probability of each sample is given by $P(z_1...z_p, e_1...e_m) = \left[\prod_i^p P(z_i|Parents(z_i))\right] \cdot [\prod_i^m P(e_i|Parents(e_i))]$. The pseudocode for Likelihood Weigting is provided below.

function LIKELIHOOD-WEIGHTING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
    N, the total number of samples to be generated
  local variables: **W**, a vector of weighted counts for each value of X, initially zero

  for j = 1 to N **do**
    x, $w \leftarrow$ WEIGHTED-SAMPLE(bn, **e**)
    $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where x is the value of X in **x**
  return NORMALIZE(**W**)

---

function WEIGHTED-SAMPLE(bn, **e**) **returns** an event and a weight

  $w \leftarrow 1$; **x** $\leftarrow$ an event with n elements initialized from **e**
  foreach variable X_i **in** $X_1, \ldots, X_n$ **do**
    if X_i is an evidence variable with value x_i in **e**
      then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
      else **x**$[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return **x**, w

Figure 14.15 The likelihood-weighting algorithm for inference in Bayesian networks. In WEIGHTED-SAMPLE, each nonevidence variable is sampled according to the conditional distribution given the values already sampled for the variable's parents, while a weight is accumulated based on the likelihood for each evidence variable.

For all three of our sampling methods (prior sampling, rejection sampling, and likelihod weighting), we can get increasing amounts of accuracy by generating additional samples. However, of the three, likelihood weighting is the most computationally efficient, for reasons beyond the scope of this course.

Gibbs Sampling

Gibbs Sampling is a fourth approach for sampling. In this approach, we first set all variables to some totally random value (not taking into account any CPTs). We then repeatedly pick one variable at a time, clear its value, and resample it given the values currently assigned to all other variables.

For the T, C, S, E example above, we might assign t = true, c = true, s = false, and e = true. We then pick one of our four variables to resample, say S, and clear it. We then pick a new variable from the distribution $P(S|+t,+c,+e)$. This requires us knowing this conditional distribution. It turns out that we can easily compute the distribution of any single variable given all other variables. More specifically, $P(S|T,C,E)$ can be calculated only using the CPTs that connect S with its neighbors. Thus, in a typical Bayes Net, where most variables have only a small number of neighbors, we can precompute the conditional distributions for each variable given all of its neighbors in linear time.

We will not prove this, but if we repeat this process enough times, our later samples will eventually converge to the correct distribution even though we may start from a low-probability assignment of values. If you're curious, there are some caveats beyond the scope of the course that you can read about under the Failure Modes section of the Wikipedia article for Gibbs Sampling.

The pseudocode for Gibbs Sampling is provided below.

function GIBBS-ASK(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  local variables: **N**, a vector of counts for each value of X, initially zero
    Z, the nonevidence variables in bn
    x, the current state of the network, initially copied from **e**

  initialize **x** with random values for the variables in **Z**
  for j = 1 to N **do**
    for each Z_i in **Z do**
      set the value of Z_i in **x** by sampling from $\mathbf{P}(Z_i|mb(Z_i))$
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in **x**
  return NORMALIZE(**N**)

Figure 14.16 The Gibbs sampling algorithm for approximate inference in Bayesian networks; this version cycles through the variables, but choosing variables at random also works.

Conclusion

To summarize, Bayes' Nets is a powerful representation of joint probability distributions. Its topological structure encodes independence and conditional independence relationships, and we can use it to model arbitrary distributions to perform inference and sampling.

In this note, we covered two approaches to probabilistic inference: exact inference and probabilistic inference (sampling). In exact inference, we are guaranteed the exact correct probability, but the amount of computation may be prohibitive.

The exact inference algorithms covered were:

- Inference By Enumeration
- Variable Elimination

We can turn to sampling to approximate solutions while using less compute.

The sampling algorithms covered were:

- Prior Sampling
- Rejection Sampling
- Likelihood Weighting
- Gibbs Sampling

D-Separation (Optional)

One useful question to ask about a set of random variables is whether or not one variable is independent from another, or if one random variable is conditionally independent of another given a third random variable. Bayes' Nets representation of joint probability distributions gives us a way to quickly answer such questions by inspecting the topological structure of the graph.

We already mentioned that **a node is conditionally independent of all its ancestor nodes in the graph given all of its parents.**

We will present all three canonical cases of connected three-node two-edge Bayes' Nets, or triples, and the conditional independence relationships they express.

Causal Chains

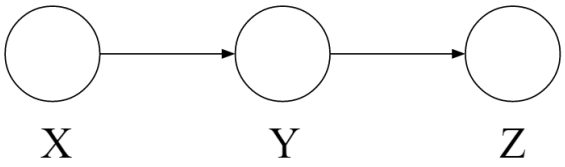


Figure 1: Causal Chain with no observations.

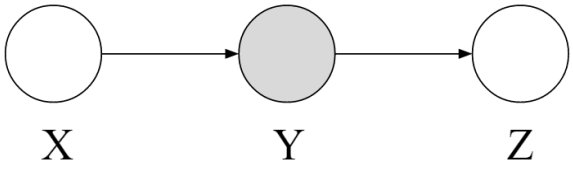


Figure 2: Causal Chain with Y observed.

Figure 1 is a configuration of three nodes known as a **causal chain**. It expresses the following representation of the joint distribution over X, Y, and Z:

$$P(x,y,z) = P(z|y)P(y|x)P(x)$$

It's important to note that X and Z are not guaranteed to be independent, as shown by the following counterexample:

$$P(y|x) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{else} \end{cases} \qquad P(z|y) = \begin{cases} 1 & \text{if } z = y \\ 0 & \text{else} \end{cases}$$

In this case, $P(z|x) = 1$ if $x = z$ and 0 otherwise, so X and Z are not independent.

However, we can make the statement that $X \perp\!\!\!\perp Z|Y$, as in Figure 2. Recall that this conditional indepdence means:

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

We can prove this statement as follows:

$$\begin{aligned} P(X|Z,y) &= \frac{P(X,Z,y)}{P(Z,y)} = \frac{P(Z|y)P(y|X)P(X)}{\sum_x P(X,y,Z)} = \frac{P(Z|y)P(y|X)P(X)}{P(Z|y)\sum_x P(y|x)P(x)} \\ &= \frac{P(y|X)P(X)}{\sum_x P(y|x)P(x)} = \frac{P(y|X)P(X)}{P(y)} = P(X|y) \end{aligned}$$

An analogous proof can be used to show the same thing for the case where X has multiple parents. To summarize, in the causal chain chain configuration, $X \perp\!\!\!\perp Z|Y$.

Common Cause

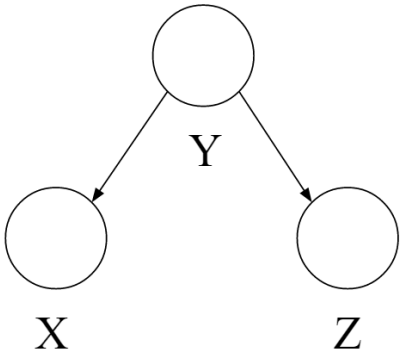


Figure 3: Common Cause with no observations.

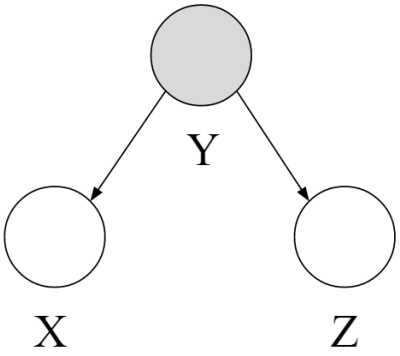


Figure 4: Common Cause with Y observed.

Another possible configuration for a triple is the **common cause**. It expresses the following representation:

$$P(x,y,z) = P(x|y)P(z|y)P(y)$$

Just like with causal chain, we can show that X is not guaranteed to be independent of Z with the following counterexample distribution:

$$P(x|y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{else} \end{cases} \qquad P(z|y) = \begin{cases} 1 & \text{if } z = y \\ 0 & \text{else} \end{cases}$$

Then $P(x|z) = 1$ if $x = z$ and 0 otherwise, so X and Z are not independent.

But it is true that $X \perp\!\!\!\perp Z|Y$. That is, X and Z are independent if Y is observed as in Figure 4. We can show this as follows:

$$P(X|Z,y) = \frac{P(X,Z,y)}{P(Z,y)} = \frac{P(X|y)P(Z|y)P(y)}{P(Z|y)P(y)} = P(X|y)$$

Common Effect

The final possible configuration for a triple is the **common effect**, as shown in the figures below.

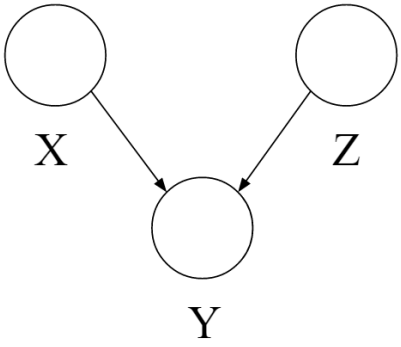


Figure 5: Common Effect with no observations.

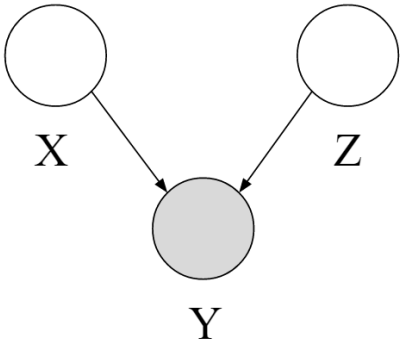


Figure 6: Common Effect with Y observed.

It expresses the representation:

$$P(x,y,z) = P(y|x,z)P(x)P(z)$$

In the configuration shown in Figure 5, X and Z are independent: $X \perp\!\!\!\perp Z$. However, they are not necessarily independent when conditioned on Y (Figure 6). As an example, suppose all three are binary variables. X and Z are true and false with equal probability:

$$\begin{aligned} P(X = true) &= P(X = false) = 0.5 \\ P(Z = true) &= P(Z = false) = 0.5 \end{aligned}$$

and Y is determined by whether X and Z have the same value:

$$P(Y|X,Z) = \begin{cases} 1 & \text{if } X = Z \text{ and } Y = true \\ 1 & \text{if } X \neq Z \text{ and } Y = false \\ 0 & \text{else} \end{cases}$$

Then X and Z are independent if Y is unobserved. But if Y is observed, then knowing X will tell us the value of Z, and vice-versa. So X and Z are *not* conditionally independent given Y.

Common Effect can be viewed as "opposite" to Causal Chains and Common Cause – X and Z are guaranteed to be independent if Y is not conditioned on. But when conditioned on Y, X and Z may be dependent depending on the specific probability values for $P(Y|X,Z)$).

This same logic applies when conditioning on descendants of Y in the graph. If one of Y's descendant nodes is observed, as in Figure 7, X and Z are not guaranteed to be independent.

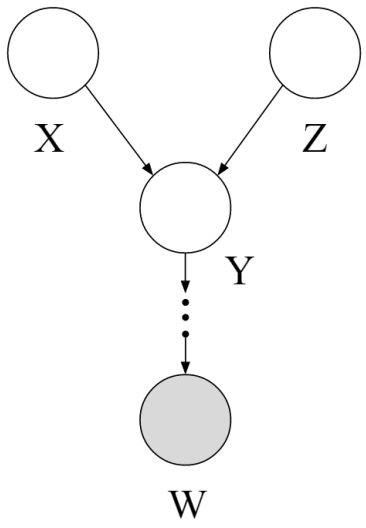


Figure 7: Common Effect with child observations.

General Case, and D-separation

We can use the previous three cases as building blocks to help us answer conditional independence questions on an arbitrary Bayes' Net with more than three nodes and two edges. We formulate the problem as follows:

Given a Bayes Net G, two nodes X and Y, and a (possibly empty) set of nodes $\{Z_1, \ldots Z_k\}$ that represent observed variables, must the following statement be true: $X \perp\!\!\!\perp Y | \{Z_1, \ldots Z_k\}$?

D-separation (directed separation) is a property of the structure of the Bayes Net graph that implies this conditional independence relationship, and generalizes the cases we've seen above. If a set of variables $Z_1, \cdots Z_k$ d-separates X and Y, then $X \perp\!\!\!\perp Y | \{Z_1, \cdots Z_k\}$ in all possible distributions that can be encoded by the Bayes net.

We start with an algorithm that is based on a notion of reachability from node X to node Y. (**Note: this algorithm is not quite correct! We'll see how to fix it in a moment.**)

1. Shade all observed nodes $\{Z_1, \ldots Z_k\}$ in the graph.
2. If there exists an undirected path from X and Y that is not blocked by a shaded node, X and Y are "connected".
3. If X and Y are connected, they're not conditionally independent given $\{Z_1, \ldots Z_k\}$. Otherwise, they are.

However, this algorithm only works if the Bayes' Net has no Common Effect structure within the graph, because if it exists, then two nodes are "reachable" when the Y node in Common Effect is activated (observed). To adjust for this, we arrive at the following **d-separation algorithm**:

1. Shade all observed nodes $\{Z_1, \ldots, Z_k\}$ in the graph.
2. Enumerate all undirected paths from X to Y.
3. For each path:
   (a) Decompose the path into triples (segments of 3 nodes).
   (b) If all triples are active, this path is active and *d-connects* X to Y.
4. If no path d-connects X and Y, then X and Y are d-separated, so they are conditionally independent given $\{Z_1, \ldots, Z_k\}$

Any path in a graph from X to Y can be decomposed into a set of 3 consecutive nodes and 2 edges - each of which is called a triple. A triple is active or inactive depending on whether or not the middle node is observed. If all triples in a path are active, then the path is active and *d-connects* X to Y, meaning X is not guaranteed to be conditionally independent of Y given the observed nodes. If all paths from X to Y are inactive, then X and Y are conditionally independent given the observed nodes.

Active triples: We can enumerate all possibilities of active and inactive triples using the three canonical graphs we presented below in Figure 8 and 9.

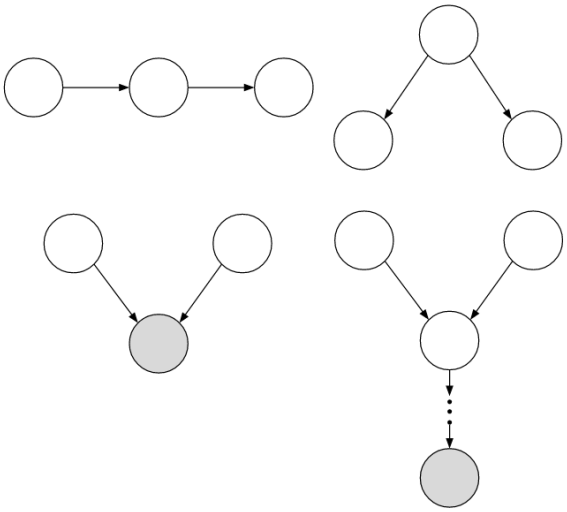


Figure 8: Active triples

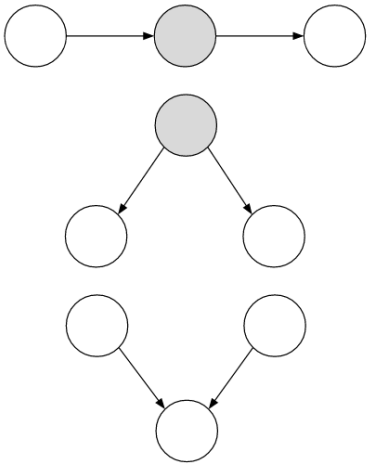
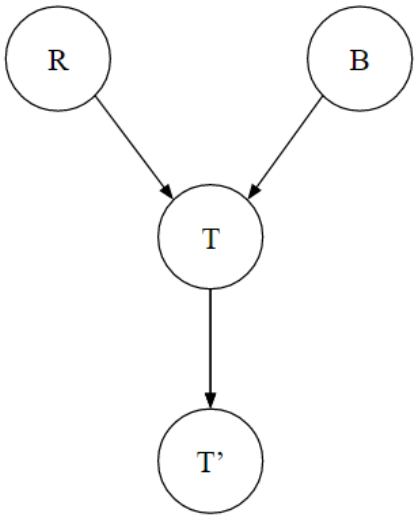


Figure 9: Inactive triples

Examples

Here are some examples of applying the d-separation algorithm:



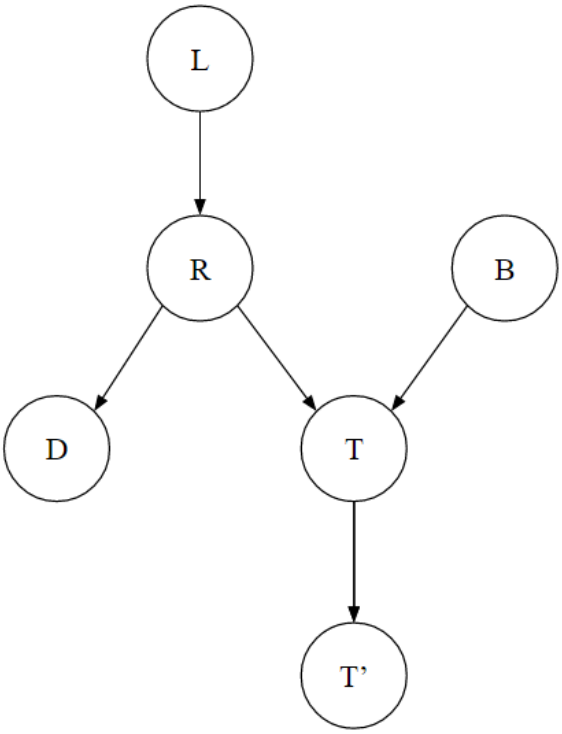
This graph contains the common effect and causual chain canonical graphs.

a) $R \perp\!\!\!\perp B$ – Guaranteed

b) $R \perp\!\!\!\perp B|T$ – Not guaranteed

c) $R \perp\!\!\!\perp B|T'$ – Not guaranteed

d) $R \perp\!\!\!\perp T'|T$ – Guaranteed



This graph contains combinations of all three canonical graphs (can you list them all?).

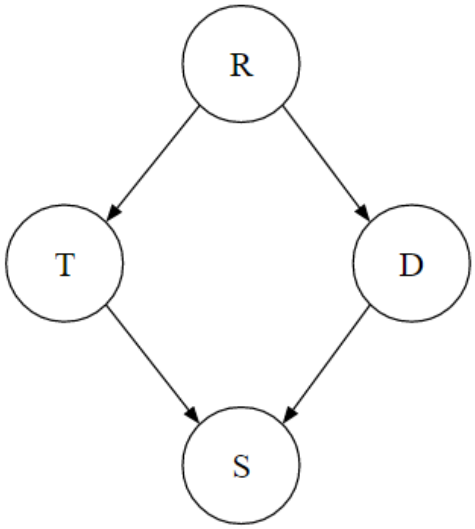
a) $L \perp\!\!\!\perp T' | T$ – Guaranteed

b) $L \perp\!\!\!\perp B$ – Guaranteed

c) $L \perp\!\!\!\perp B | T$ – Not guaranteed

d) $L \perp\!\!\!\perp B | T'$ – Not guaranteed

e) $L \perp\!\!\!\perp B | T, R$ – Guaranteed



This graph contains combinations of all three canonical graphs.

a) $T \perp\!\!\!\perp D$ – Not guaranteed

b) $T \perp\!\!\!\perp D | R$ – Guaranteed

c) $T \perp\!\!\!\perp D | R, S$ – Not guaranteed