

Lecture 9

Probabilistic Reasoning

Probability is important in many branches of mathematics and science, and AI is not an exception. Many tasks require us to evaluate probabilities.

This lecture explains the basic probability framework, discuss a representation which succinctly describe probabilistic knowledge, and show how probabilistic reasoning can take place.

Reference:

- Textbook Chapter 13–15.

AI(0270)

Sources of uncertainty

So instead of saying the very long sentence, we would like to write
 $Symptom(p, Toothache) \Rightarrow Disease(p, Cavity) \vee \dots \vee RareEvent(p)$

But now we have a difficulty: **we never know that a rare event occurs!**

We encounter uncertainty in such systems because...

- we are **lazy** to write a rule that is completely correct, and even if we do, the result will be so **large** that it becomes unusable.
- we are **ignorant** about some of the causes and consequence relationship, so we can only **approximate** the physical world.
- we cannot administer all the **tests** required to obtain all information to give a completely correct inference. Doing so would be either impossible or too expensive.

AI(0270)-9.2

Why probability is important in AI?

We need probabilistic reasoning in nearly all realistic AI (or scientific) applications. We have seen that some games are probabilistic. Some other examples...

- If you'd just have a sensor to detect the robot, it is possible that sometimes the sensor misbehaves, and you never know when.
- If you'd like to understand a natural language sentence, there are a lot of ambiguity and you must use contextual and common-sense to choose the most probable one.
- For a speech recognition system (turning sounds to words), usually the sound of the speaker has a little distortion, and your only hope to acquire human-like accuracy is to use sounds around to guess what is the most likely word spoken.
- Even in domains as simple as the Wumpus world, only probability can guide us about what to do when there is no "safe" way out.

AI(0270)-9.4

Using probability to model uncertainty

Suppose we want to have a system to diagnose dental problems.

- We would like to have a rule like this:
 $Symptom(p, Toothache) \Rightarrow Disease(p, Cavity)$
- But we **can't**. There are **many other reasons to have toothache**, and just having toothache is not enough for deducing cavity.
- So we are forced to write something like...
 $Symptom(p, Toothache) \Rightarrow Disease(p, Cavity) \vee Disease(p, GumDisease) \vee Disease(p, ImpactedWisdom) \vee \dots$
- But... it is difficult, since there are really so many reasons.
- How about 'causal rules', e.g., $Disease(p, Cavity) \Rightarrow Symptom(p, Toothache)$?
- Still incorrect (although much better), and difficult to use.

AI(0270)-9.1

- Probability solves the problem by using a number to **summarize** all the uncertainty we meet.
- So we might say that there is a 80% chance that a patient has cavity if he has a toothache. The 20% summarizes all the other possibilities.
- In Propositional (and First-Order) Logic, everything is either true or false. It is still the case in our probabilistic world.
- In Propositional (and First-Order) Logic, if we don't think something true or false, we have to say we "don't know anything about it".
- When doing probabilistic reasoning, we want to be able to believe something "is likely to be true" or "is likely to be false". So it **extends** FOL.
 We use only propositions—using First-Order statements with probability is quite complicated.

AI(0270)-9.3

What the probabilities means?

Why we say that 80%, rather than 75% or 5%?

- By 80%, we mean that **if similar things happen many times**, in 80% of the cases the patient has cavity.
- Such knowledge can be collected **by using a large database of past experience**, or **combined** from different sources of information.
- In cases that past experience is not available or is too small, we might need to ask a domain expert to **supply a subjective estimate**.

But be careful...

- A probability makes sense **only if we know what are the evidences!**
- E.g., we might say that the probability of a random people to have cavity to be 1/2, but if we add the evidence that he go to the dentist or that he complains about toothache, it is no longer 1/2!

AI(0270)-9.5

Notation: Conditional Probability

This poses a difficulty: we **cannot** simply say that “the probability to have cavity is $1/2$ ”. We have to be more specific about the evidences. We use the following notation:

- $P(\text{Cavity}=\text{true} \mid \text{Toothache}=\text{true})$: the **conditional probability** that the propositional sentence *Cavity* is true, given that *Toothache* is true.
- We will abbreviate it as $P(\text{cavity} \mid \text{toothache})$.
- In probability theory, *Cavity* is said to be a **random variable**, and that $\text{Cavity} = \text{true}$ (or *false*) is said to be an **event**.
- If we have no evidence, then the evidence part must be trivially true, e.g., $P(\text{cavity} \mid \text{true})$.
- In this case, we write it as $P(\text{cavity})$ —the **prior probability** that $\text{Cavity} = \text{true}$ holds (i.e., when no evidence is known).
In contrast, $P(\text{cavity} \mid \text{toothache})$ is said to be posterior probability.

AI(0270)-9.6

How probabilities work

In principle, to work with probability theory, we only need three axioms and one definition:

- For any proposition A , $0 \leq P(A) \leq 1$
- $P(\text{True}) = 1$, $P(\text{False}) = 0$
- Probability of disjunction is given by
 $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
This is just the Venn diagram.
- $P(A \mid B) = P(A \wedge B) / P(B)$
E.g., if $P(A \wedge B) = 0.4$ and $P(B) = 0.5$, then $P(A \mid B) = 0.8$. Intuitively, if there are 100 samples, $A \wedge B$ holds in 40 of them, and B holds in 50 of them, and we already know B occurs, then we expect to find A in 80% of these samples.

AI(0270)-9.7

Probability distribution

- For a random variable with multiple values, we usually have to know the probability of all the possible values.
- E.g., $P(\text{Weather}=\text{sunny}) = 0.3$, $P(\text{Weather}=\text{cloudy}) = 0.5$, $P(\text{Weather}=\text{raining}) = 0.2$.
They must sum to 1, due to the probability axioms.
- We say such a collection to be a **probability distribution**, written as $\mathbf{P}(\text{Weather})$. (Note the bold P.)
- It is usually written as a vector: $\mathbf{P}(\text{Weather}) = (0.3, 0.5, 0.2)$.
- We can then say something like $\mathbf{P}(\text{Weather} \mid \text{Prediction}) = \mathbf{P}(\text{Weather} \wedge \text{Prediction}) / \mathbf{P}(\text{Prediction})$.
This represent a lot of sentences, not just one. Things like $P(\text{Weather}=\text{cloudy} \mid \text{Prediction}=\text{raining}) = P(\text{Weather}=\text{cloudy} \wedge \text{Prediction}=\text{raining}) / P(\text{Prediction}=\text{raining})$.

AI(0270)-9.8

Joint probability distributions

- Random variables can be more complicated. It can, for example, be a tuple containing many values (e.g., *Weather* and *Prediction*).
- A probability distribution of the tuple containing *Weather* and *Prediction* is said to be the **joint distribution** of the two variables.
- The joint will have values like $P(V=\{\text{sunny}, \text{sunny}\}) = 0.2$, $P(V=\{\text{sunny}, \text{cloudy}\}) = 0.05$, etc.
It is usually expressed like a table. See the next page.
- In a special case, a tuple can contain **all** the simple random variables in our domain. Its distribution is said to be a **full joint probability distribution**.
- A full joint probability distribution contains the probability that **each possible combination of events** occur.

AI(0270)-9.9

A full joint is all we need

If we have the full joint, we can find any conditional probability. E.g., if

	<i>toothache</i>	\neg <i>toothache</i>
<i>cavity</i>	0.04	0.06
\neg <i>cavity</i>	0.01	0.89

Then $P(\text{cavity}) = 0.04 + 0.06 = 0.1$.

The axiom says that we should subtract $P(\text{cavity} \wedge \neg \text{cavity})$, but that value is 0 because $\text{cavity} \wedge \neg \text{cavity} = \text{false}$ and $P(\text{false}) = 0$.

Another example: $P(\neg \text{toothache} \mid \text{cavity}) = 0.06 / 0.1 = 0.6$.

In general, to find a conditional distribution, we just **sum up all distributions** for (1) both the evidence and the variable being true, and (2) the evidence being true. The quotient is the conditional probability.

The sums are called “marginal probabilities”, as they are usually written on the margins of tables.

AI(0270)-9.10

So a big joint is enough for anything we need...

Unluckily, it is usually infeasible to ask for the full joint...

- Assume we have n boolean variables. Then we have 2^n numbers to specify!
So we can handle no more than 30 boolean random variables!
- And... **we have no way to estimate their values!**
When there are 30 boolean variables, full-joint probabilities will be on average as small as $1 / 2^{30} \approx 10^{-9}$ (and usually many are much smaller). We need 10^{10} experiments to have a good idea about their values!

A fully specified joint is **too big a requirement** for our reasoning system. We have to do with a much simpler system that can be obtained and stored more easily.

AI(0270)-9.11

Normalization

Before continuing, let's look at a few facts about joint probabilities and conditional probabilities first, as they will be used repeatedly.

- Suppose we want to find $P(A | b)$. By definition, this is the same as $P(A \wedge b)/P(b)$.
- If we find it using the joint, we will have to get $P(A \wedge b)$. But this also gives enough information to get $P(b)$, as it is simply the **sum** of all $P(A \wedge b)$.
- We can do it another way: just compute $P(A \wedge b)$. This gives us a vector of probabilities.
- We can then **normalize** the vector of probabilities so that it sums to 1. This operation, if done now, actually finds $P(b)$.
- But the normalization can be delayed further if we have some more constants to divide, which thus can save time.

AI(0270)-9.12

Bayes' Rule

So instead of evaluating $P(A | b)$, we can evaluate $P(A \wedge b)$ and normalize. But we can take one further step.

- By definition, $P(b | A) = P(A \wedge b)/P(A)$.
- We can write it in reverse, i.e., $P(A \wedge b) = P(b | A) \times P(A)$.

So instead of finding $P(A | b)$, we find $P(b | A) \times P(A)$ and normalize. This is so useful that it is named, after the one inventing it (Bayes' rule).

Useful because it turns causal probabilities to diagnostic ones and vice-versa.

One final note: all such rules work when we add some additional knowledge to every term. E.g., we know

$$P(A | b, c, d) = P(A \wedge b | c, d)/P(b | c, d).$$

I.e., we can find $P(A \wedge b | c, d)$ and normalize.

AI(0270)-9.13

Independence

In most real-world problems, the full joint probabilities can be found (or approximated) by specifying much fewer numbers. The key ideas are **independence** and **conditional independence**.

- We say that a proposition A is **independent** of another proposition B if we have $P(A | B) = P(A)$
- Intuitively, this means that **the probability of A does not change when we acquire the knowledge about B** .
- With the definition of $P(A | B)$, we can write it as $P(A \wedge B) = P(A)P(B)$, which is usually called the **product rule**.
- It is clear that if A is independent of B , then B is independent of A .
- Similarly, we say that two random variables X and Y are independent if $P(X | Y) = P(X)$.
I.e., $P(X=x | Y=y) = P(X=x)$ for all x and y .

AI(0270)-9.14

Conditional independence

Sometimes two variables or events are **not independent** by themselves, but if we **have the knowledge of another variable** or event, then they **become independent**.

- We say two events A and B are independent conditioned on C if $P(A | B \wedge C) = P(A | C)$.
Once we know C , then the knowledge of B does not affect the probability of A .
- E.g., suppose the dentist uses a probe to check whether there is cavity. We get an event *catch* if the probe is caught by a glitch on a tooth.
- The events *catch* and *toothache* are **not independent**—it is more likely for a tooth to catch the probe if that person has toothache.
- But given *cavity*, they might become independent!
I.e., $P(\text{toothache} | \text{cavity} \wedge \text{catch}) = P(\text{toothache} | \text{cavity})$. Why? *Toothache* occurs when cavity affect the nerves in the tooth, *Catch* occurs when the dentist (luckily) probe a point that the cavity starts. These are independent processes.

AI(0270)-9.15

How conditional independence helps?

Assume that *toothache* and *catch* are independent given *cavity*...

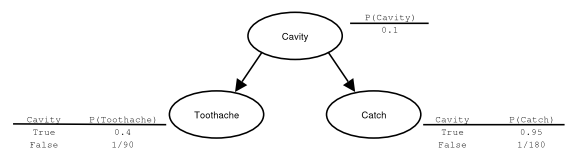
- Let's do a little bit of probability manipulations:
$$\begin{aligned} P(\text{Cavity} \wedge \text{Toothache} \wedge \text{Catch}) &= P(\text{Toothache} | \text{Cavity} \wedge \text{Catch}) \times P(\text{Cavity} \wedge \text{Catch}) \\ &= P(\text{Toothache} | \text{Cavity}) \times P(\text{Cavity} \wedge \text{Catch}) \\ &= P(\text{Toothache} | \text{Cavity}) \times P(\text{Catch} | \text{Cavity}) \times P(\text{Cavity}) \end{aligned}$$
- Since $P(A | B)$ needs only 2 values to be specified, we only need to specify 5 values instead of 7 in $P(\text{Cavity} \wedge \text{Toothache} \wedge \text{Catch})$.
- And... **the probabilities we need are causal probability**, i.e., probability of a consequence given the cause—which are much easier to estimate and less likely to change!
Because causal probabilities directly reflect the mechanisms in which the world work, e.g., how cavity causes toothache.

AI(0270)-9.16

Belief networks

But there is one more thing to store: we need to know the conditional independence relation before we can calculate the joint that way.

We need a way to specify it—**belief network**. E.g.,



The numbers are those that are required to represent $P(\text{Cavity})$, $P(\text{Toothache} | \text{Cavity})$ and $P(\text{Catch} | \text{Cavity})$. We call them **Conditional Probability Tables (CPTs)**.

What are the edges? Intuitively, *Cavity* is the primary cause of *Toothache*, and *Cavity* is the primary cause of *Catch*.

AI(0270)-9.17

The full joint view of belief network

What does the graph mean exactly?

- We can view the network as a specification the **full joint**.
- The value of a full joint (i.e., given the values of all variables) is given by **multiplying** all the corresponding probabilities in the graph. E.g.,

$$P(\text{cavity} \wedge \text{toothache} \wedge \neg \text{catch}) \\ = 0.1 \times 0.4 \times (1 - 0.95) \\ = 0.002$$

It can easily be seen that if we adds up all the values obtained by considering different combinations, we get 1.

- This “full-joint view” allows us to **use** the belief network, we now have the full joint and can compute all conditional probabilities we ever need.
- But it doesn’t directly answer us “what are the conditional independence it represents”.

AI(0270)-9.18

Reconciliation with the definition of conditional probabilities

- Suppose we list out the random variables in a way that **the parents are listed before the children**.
- E.g., in our network, Cavity, Toothache, Catch.
- By repeatedly applying the “product rule” (i.e., definition of conditional probability), we know

$$P(\text{Cavity} \wedge \text{Toothache} \wedge \text{Catch}) \\ = P(\text{Cavity})P(\text{Toothache} | \text{Cavity})P(\text{Catch} | \text{Toothache} \wedge \text{Cavity})$$

- The full joint view tells us that

$$P(\text{Cavity} \wedge \text{Toothache} \wedge \text{Catch}) \\ = P(\text{Cavity})P(\text{Toothache} | \text{Cavity})P(\text{Catch} | \text{Cavity})$$

- So $P(\text{Catch} | \text{Toothache} \wedge \text{Cavity}) = P(\text{Catch} | \text{Cavity})$.
Toothache is independent on Catch given Cavity, as before.

AI(0270)-9.19

Conditional independence view

In general...

- If V_1, V_2, \dots, V_n are the variables in the network, in an order in which an arrow never points from V_i to V_j when $i > j$,
- then V_i is independent on $V_j (i > j)$ conditioned on all parents of V_i .

In words, given all parents of a variable V_i , then V_i is independent on all its (other) predecessors.

In some sense, the parents of V_i separate all variables other than the descendants of V_i from V_i , so they become independent.

We call this the **conditional independence view** of the belief network. It is equivalent to the full joint view.

AI(0270)-9.20

Building a belief network

Now that we understand what belief network really means, we can think about how to **build** a belief network. The steps:

- Choose the set of relevant random **variables** to describe the domain.
- Choose an **ordering** X_1, X_2, \dots, X_n of the variables.
- for $i = 1$ to n , do the following:
 - Add a node X_i to the network.
 - Find a minimal subset S of X_1, X_2, \dots, X_{i-1} such that $P(X_i | X_1, X_2, \dots, X_{i-1}) = P(X_i | S)$.
 - Add edges from X_i to each element in S .
 - Define the CPT for variable X_i .

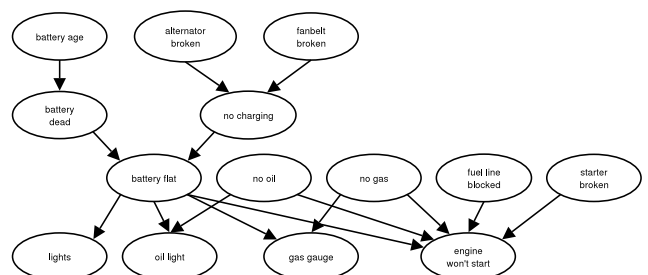
AI(0270)-9.21

Choosing an ordering

- There is something quite arbitrary: **how to choose an ordering?**
- In general, we want to choose an ordering in which **causes run before consequences**, so that step 3b results in the smallest S .
- The procedure **produces a correct network even if we choose a bad ordering**. But we would then get a larger network (i.e., more edges).
- We don’t like large networks**: CPT sizes **increase exponentially** with the number of incoming edges, and the **time cost** of asking for posterior probabilities **increase likewise**.

AI(0270)-9.22

A typical network: car problem



Note that there are usually some nodes, like no charging, battery dead and battery flat, which is solely to **reduce the network size**.

Otherwise, every consequence has to depend on battery age, alternater broken and fanbelt broken.

AI(0270)-9.23

Computing conditional probabilities

How to use a belief network to compute, e.g., $P(A | B)$?

- For each combination of variables such that both A and B are true, multiply all the values of the CPT to get the full joint probability.
- Sum up all the products.
- Do the same thing for all combinations that B is true.
- Divide the two sums, and we get the conditional probability we want.

But... there is a **huge** number of combinations of variables! If we have 30 boolean variables, we have $2^{30} = 10^9$ of them...

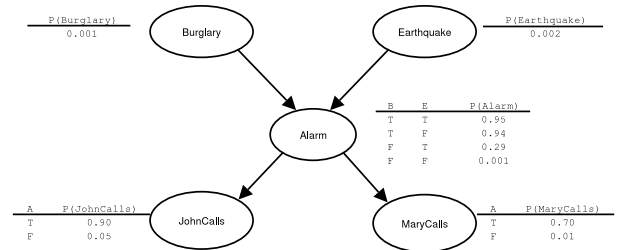
Belief network helps us to reduce the storage requirement. But can it also help us reduce the computational time?

The belief network we've seen is a bit too simple to illustrate the ideas. We'll use a slightly more complex example.

AI(0270)-9.24

Example

- One author of the book has installed a burglary alarm, and have two neighbours to phone him if they hear the alarm.
- But he lives at a place where there are occasional earthquake, which can also trigger the alarm.



Note that the Alarm CPT has 4 entries to make up all combinations.

AI(0270)-9.25

A slow way to calculate conditional probabilities

Given both calls, what is the chance that burglary happened?

- So we want the probability distribution $\mathbf{P}(\text{Burglary} | \text{JohnCall}=\text{true} \wedge \text{MaryCall}=\text{true})$. (We will abbreviate it as $\mathbf{P}(B | j, m)$).
We use commas instead of \wedge to save space.
- We can find $\mathbf{P}(B, j, m)$ and normalize. But how to find the (partial) joint, $\mathbf{P}(B, j, m)$?
- The full joint probabilities have two more variables, E and A . So we simply sum over all possibilities. I.e., $\mathbf{P}(B, j, m) = \sum_{A, E} \mathbf{P}(B, j, m, E, A)$
- So we can start with the full joint values computed by the belief network, add them up in particular ways, and find the conditional probability we want.

AI(0270)-9.26

The actual calculation

- $P(b, e, a, j, m) = P(b)P(e)P(a | b, e)P(j | a)P(m | a)$
 $P(b, e, \neg a, j, m) = P(b)P(e)P(\neg a | b, e)P(j | \neg a)P(m | \neg a)$
 $P(b, \neg e, a, j, m) = P(b)P(\neg e)P(a | b, \neg e)P(j | a)P(m | a)$
 $P(b, \neg e, \neg a, j, m) = P(b)P(\neg e)P(\neg a | b, \neg e)P(j | \neg a)P(m | \neg a)$
 All values on the right are entries in one of the CPTs of the belief network.
- They have values 0.00000119700, 0.00000000005, 0.00059101560 and .00000002994 respectively. Summing them up we get $P(b, j, m) = 0.00059224259$.
- Similarly, $P(\neg b, j, m) = 0.001491857649$.
- Normalizing, we get $P(b | j, m) = 0.284$. (So small!!!)

AI(0270)-9.27

How to speed it up?

Note that a lot of lookups and computations are **repeated** if we are not careful.

Why we need to lookup twice for each of $P(j | a)$, $P(m | a)$, $P(\neg j | a)$, $P(\neg m | a)$, $P(b)$ and $P(e)$? Even worse, in more complex networks they might not be a simple lookup (but instead a long computation)!

We can group terms out to achieve some of the savings. E.g.,

$$\begin{aligned}
 P(b) & (P(e) (P(a | b, e)P(j | a)P(m | a) \\
 & + P(\neg a | b, e)P(j | \neg a)P(m | \neg a)) \\
 + P(\neg e) & (P(a | b, \neg e)P(j | a)P(m | a) \\
 & + P(\neg a | b, \neg e)P(j | \neg a)P(m | \neg a)))
 \end{aligned}$$

Still, $P(j | a)$, $P(m | a)$, $P(j | \neg a)$ and $P(m | \neg a)$ are repeated. Some of you might yell out, "dynamic programming!"

AI(0270)-9.28

Let's formalize it...

- What we want to compute:

$$\sum_{E, A} P(b)P(E)P(A | b, E)P(j | A)P(m | A).$$
- Moving unrelated variables outside sums, we get

$$P(b) \sum_E P(E) \sum_A P(A | b, E)P(j | A)P(m | A).$$
- Now we will compute things from the **right**:
 - Find $P(m | A)$ for all possibilities of A . Call it $\mathbf{f}_M(A)$.
It is a vector. E.g., here it is $(a : 0.7, \neg a : 0.01)$.
 - Find $P(j | A)$ for all possibilities of A . Call it $\mathbf{f}_j(A)$.
 - Find $P(A | b, E)$ for all possibilities of A and E . Call it $\mathbf{f}_A(A, E)$.
Suffix is part of sum that we have considered, arguments are those variables that are not fixed.

AI(0270)-9.29

Let's formalize it...

- We multiply $f_M(A)$, $f_J(A)$ and $f_A(E)$ for each combination of A and E . Call the result $f_{AJM}(A, E)$.
- We call this **pointwise multiplication**: an independent product is calculated for every combination of values of the unknown variables.
- Find $\sum_A P(A | b, E)P(j | A)P(m | A)$ from $f_{AJM}(E)$; by summing all values for A . Call the result $f_{AJM}(E)$.
- We call this **summing out** the f values involving A .
- We continue by finding $P(E)$, denoted as $f_E(E)$.
- Then we do pointwise multiplication between $f_E(E)$ and $f_{AJM}(E)$, resulting in $f_{E AJM}(E)$.
- We then sum out E to get $f_{E AJM}()$, and multiply by $P(b)$.

AI(0270)-9.30

The variable elimination algorithm

- Set $S = []$, to be used as a set of known f value.
- For each variable V , in the **reverse ordering** of the belief network:
 - Use the CPT of V to construct an f value, which takes the values of V and its parents (except evidences) as arguments. Add it to S .
 - If V is not known and not a query variable
 - Find all f values in S taking V as argument. Remove them from S . Multiply all of them according to the combination of variable values (do **pointwise multiply**). Call the result f_1 .
 - Sum the entries of f_1 with different V but the same other variables (**sum-out** V). This results in a new f value, which is added to S .
- **Pointwise-multiply** the f values in S (in case of multiple query variables), and **normalize** the result.

AI(0270)-9.31

How fast is it?

- The critical part of the algorithm is for pointwise multiplication.
- Each pointwise multiplication has the potential to just adds up the number of arguments in the f value.
 - If this number of arguments is n , then the value contains 2^n values.
- In each sum-out, the number of arguments is reduced by 1.
- The efficiency of the algorithm depends on whether the acquisition or removal of variable is faster in the f values.
- If the belief network is a **tree**, the speed is about the same. This results in a **linear** time algorithm.
- In a general graph, the speed of acquisition is much faster at the beginning, resulting in an **exponential** time algorithm.
- But the problem is NP-hard, so sometimes we need exponential time.

AI(0270)-9.32

Approximation algorithms

- But for probabilistic reasoning, if you don't require exact answer, it can be done more efficiently. The idea: do **many experiments!**
- Let's start with finding **full-joint** probabilities. Each time we **pick a variable with values of all parents chosen**, and choose a value for it at a probability predicted by its parents.
 - A simple way is to choose variables in the order we build the belief network.
- E.g., for the burglary alarm domain, we can start choosing a value for B and E , using their prior probabilities.
 - At probability 0.999 (0.998) to choose $\neg b$ ($\neg e$), 0.001 (0.002) to choose b (e).
- Suppose we chose $\neg b$ and e . Then we choose a value for A , based on its entry in the CPT. I.e., at probability 0.29 (0.71) we choose a ($\neg a$).
- The frequency of the samples we get is then proportional to the full-joint probabilities, **in the long run**.

AI(0270)-9.33

Rejection sampling

So we can easily **sample** at exactly the probabilities as depicted by the belief network. Now we use it to find conditional probabilities.

- Do the sampling as in the last slide. If the resulting sample is inconsistent with the evidence, we drop it. E.g., if we want to find $P(B | j, m)$, drop the sample if it has $\neg j$ or $\neg m$.
- When each sample is generated (and not dropped), we count it towards a value in the queried variable(s). E.g., if we want to find $P(B | j, m)$, keep a count for b and another for $\neg b$.
- At the end these counts are normalized to a set of conditional probabilities. E.g., if we count 10 for b and 20 for $\neg b$, we conclude $P(B | j, m) = \{b : 0.333, \neg b : 0.667\}$.

AI(0270)-9.34

Algorithm

```

def RejectionSampling(query, evidence, num_samples, belief_net):
    count = [0, 0] # assume one boolean query variable
    for i in range(num_samples): # repeat _num_samples_ times
        x = GenSample(belief_net)
        if (not Consistent(x, evidence)):
            continue
        count[x[query]] += 1
    return Normalize(count)

def GenSample(belief_net):
    sample = []
    for i in range(belief_net.num_var):
        # Find the CPT entry based on the previous sampled value
        prob = FindProb(belief_net.cpt[i], sample)
        val = (random() < prob) # True ==> the variable is set
        sample.append(val)
    return sample

```

AI(0270)-9.35

Likelihood weighting

- There is a critical problem of rejection sampling—we simply drop too many samples!
- E.g., since the actual probability of $j \wedge m$ is 0.0021, we can expect to drop 998 experiments in 1000 of them. The algorithm is not fast at all.
- One way to deal with the problem: always choose the consistent value. E.g., when “choosing” an evidence variable J , we always choose j .
- We give the experiment a **weight** that depend on the probability of such “forced” variables. E.g., if we choose a at the third step, then to have j , m , the weighting is $0.9 \times 0.7 = 0.63$. If instead it is $\neg a$, it get a weight of 0.005.
- At the end we **sum the weights**, instead of just counting.
- This technique is called **likelihood weighting**. This solves the problem of not being able to get any sample, but...

AI(0270)-9.36

Algorithm

```
def LikelihoodWeighting(query, evidence, num_samples, belief_net):
    weights = [0, 0]
    for i in range(num_samples):
        x, weight = GenWeightedSample(belief_net, evidence)
        weights[x[query]] += weight
    return Normalize(weights)
def GenWeightedSample(belief_net, evidence):
    sample, weight = [], 1
    for i in range(belief_net.num_var):
        if i in evidence:
            sample.append(True)
            weight *= FindProb(belief_net.cpt[i], sample)
        elif (not i) in evidence:
            sample.append(False)
            weight *= 1 - FindProb(belief_net.cpt[i], sample)
        else:
            prob = FindProb(belief_net.cpt[i], sample)
            sample.append(random() < prob)
    return sample, weight
```

AI(0270)-9.37

Problem of likelihood weighting

Suppose we have a network containing just 2 variables A and B :

- a has probability 0.9999, $\neg a$ has probability 0.0001.
- B depends on A . If a , b has probability 0.0001. If $\neg a$, b has probability 0.9999.

Suppose we want $P(A | b)$. The correct probability should be [0.5, 0.5].

Since $P(a, b) = 0.9999 \times 0.0001$, and $P(\neg a, b) = 0.0001 \times 0.9999$.

But if we run the algorithm above (say, for 2000 iterations), the resulting probability is either very large or very small!!

- If $\neg a$ is not chosen in any iteration (very likely), there is no weight for $\neg a$ at all, so the estimated probability of $P(\neg a | b)$ would be 0.
- If $\neg a$ is chosen in any iteration, then its weight will dominate: since each a weighs only 0.0001. So $P(\neg a | b)$ would be very close to 1.

AI(0270)-9.38

The MCMC algorithm

- One can argue that if we have probabilities as small as 0.0001, 2000 iterations is simply not sufficient.
- But there is one unlucky fact: probabilities of independent variables **multiplies**. So it is easy to get a real small probability—even if the CPTs do not contain it.
- Some algorithms, such as MCMC (Markov Chain Monte Carlo), are designed to combat such effects.
- Similar to rejection sampling, MCMC is based on **counting** instead of weighting. But like likelihood weighting, it **never drops** experiments.
- The framework of MCMC: start with a completely random initial state, with the values of the evidence variables stuck at the known value.
- Then, at each iteration, modify the state to another by taking **samples** on non-evidence variables, and count the queried variable.

AI(0270)-9.39

State transitions in MCMC

- Because the evidence variables are not sampled, they are always the correct value, so **no dropping** of experiment is needed.
- But how to sample? The algorithm suggested: for each variable X , find the conditional probability of each value of X assuming the **current** values of **all other variables**.
- By some (somewhat involved) arguments, this makes sure that in the long run, the probability distributions of non-evidence variables follows what predicted by the belief network.
Refer to the textbook for the proof.
- There is only one problem: how we can find the conditional probability required? Using the full joint is not particularly efficient...
- But conditional independence helps us deeply...

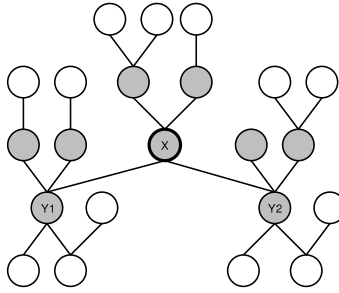
AI(0270)-9.40

Algorithm

```
def MCMC(query, evidence, num_samples, belief_net):
    curr = GenRandomState(belief_net, evidence)
    count = [0, 0]
    for i in range(num_samples):
        curr = NextSample(curr, belief_net, evidence)
        counts[x[query]] += 1
    return Normalize(counts)
def NextSample(curr, belief_net, evidence):
    sample = []
    for i in range(belief_net.num_var):
        if i in evidence:
            sample.append(True)
        elif (not i) in evidence:
            sample.append(False)
        else:
            prob = FindProbGiven(curr, belief_net.cpt[i])
            sample.append(random() < prob)
    return sample
```

AI(0270)-9.41

Markov blanket



In the figure, to find X given all other variables, we just need to consider the gray variables. All other variables (i.e., white nodes) are independent with X given the gray variables.

The computation needed is also simple: just multiply the current probability distribution of the X and the Y 's (i.e., children of X), and normalize the result.

AI(0270)-9.42

Time varying domains

- So far, all our discussion assumes that we have a **static** state and want to estimate probabilities when acquiring evidences.
- But what if our states are **dynamic**, i.e., change with time?
- Our model: the world changes with discrete **steps**. In each step the variable can take a new value, and the change may depend on previous values of the variables.
- This seems much more complex, but in some sense it is still the same conditional probability estimation problem...
- The idea: **duplicate** each variable. If our time are of value $0, 1, \dots, k$, then we duplicate the variable $k + 1$ times.
- Problem 1—how to draw the links in the belief network?
- Problem 2—network size is real large!

AI(0270)-9.43

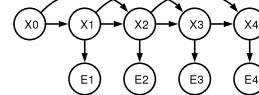
Markov chain

- There is a common special case called **Markov chain** which requires much less resources.
- The set of variables are in two primary categories: **evidence** variables which we can in general detect, and **state** variables which represent internal states.
- The evidences depend only on the state variables of the **same** step. (I.e., in the belief network, there are only arrows from the state variables of the same step.)
- The state depends only on the state variables **in a fixed, finite number of steps**. The number is called the **order** of the Markov chain.
- E.g., in a **second order** Markov chain, each state variable can depend on any state variable of the 2 previous steps.
- The dependencies may change with time.
But in most cases it doesn't change.

AI(0270)-9.44

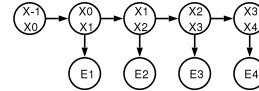
Reduction to first order

So a Markov chain looks like this:



Each circle here actually represents **multiple** variables. By clustering we may combine them to one "mega-variable".
But this is not very efficient.

We can always turn any Markov Chain to first-order by adding states. E.g., add the complete last state to the current state...



AI(0270)-9.45

Example problem

The Prisoner's room scanner:

- A prisoner want to **know whether it is raining outside**, although he has no access to the outside world.
- The **evidence** he can get: the officer comes to the prison every day, **with or without an umbrella**. The probability of having an umbrella depends on whether it rains outside.
- The prisoner also believes that whether it rains **yesterday** has an effect on whether it rains **today**.

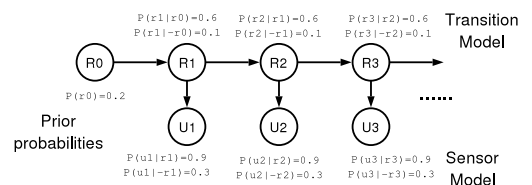
So "whether it rains" is a state variable hidden from the prisoner, and "whether the officer brings an umbrella" is an evidence variable.

AI(0270)-9.46

What probabilities we need

- The **prior probabilities** of the state variables.
- The **transition model**: the conditional probability distribution of each state variable given all state variables of the last step.
- The **sensor model**: the conditional probability distribution of each evidence variable given all state variables of this step.

E.g., in our problem, a possible set of needed knowledge:



AI(0270)-9.47

Usual tasks we want to do

Given a set of previous evidences e_1, e_2, \dots, e_t (e.g., [yes, no, no, no, yes]), we might want to know:

- **Filtering** (tracking): what is the probability distribution of the **current** state? E.g., is it likely that it rains now?
- **Prediction**: what are the probability distribution of **future** states? E.g., is it likely that it will rain tomorrow and the day after?
- **Smoothing** (hindsight): what is the probability distribution of some **past** states? E.g., is it likely that it has rained yesterday and the day before?
- **Most likely explanation**: what is the **most likely sequence** of state variables that leads to it? E.g., what is the most likely sequence of rain/not rain in these 5 days?

They can be solved by Variable Elimination, but there are more efficient ways.

AI(0270)9.48

One step prediction

- **Basic problem**: Given the probabilities of a particular step $t+k$ ($k \geq 0$), find the probability of one step further, with no additional evidence.
- I.e., find $\mathbf{P}(X_{t+k+1} | e_{1:t})$ in terms of $\mathbf{P}(X_{t+k} | e_{1:t})$.
- All we need to do: **enumerate** all possible current states and their probabilities. **Multiply** with the CPT in the **transition model** to find the joint, and **add them up** to find the needed marginal probability.
- I.e., $\mathbf{P}(X_{t+k+1} | e_{1:t}) = \sum_x \mathbf{P}(X_{t+k} = x | e_{1:t}) \mathbf{P}(X_{t+k+1} | X_{t+k} = x)$.
The rightmost term should really be $\mathbf{P}(X_{t+k+1} | X_{t+k} = x, e_{1:t})$, but the $e_{1:t}$ can be dropped because given X_{t+k} they are all independent of X_{t+k+1} .
- E.g., if we somehow know $P(r_5 | u' s) = 0.6$ and has no evidences about any U_i later than step 5, then $\hat{P}(r_6 | u' s) = 0.6 \times 0.6 + 0.4 \times 0.1 = 0.4$.
- Repeated application gives further predictions.

AI(0270)9.49

Filtering

- Filtering is needed in almost all systems implementing probabilistic reasoning. The system **keeps and updates** $\mathbf{f}_{1:t} = \mathbf{P}(X_t | e_{1:t})$.
- I.e., we can use $\mathbf{f}_{1:t}$ to find $\mathbf{f}_{1:t+1}$. How?
- Let's start with the extra evidence. $\mathbf{P}(X_{t+1} | e_{1:t+1})$ is $\mathbf{P}(X_{t+1} | e_{1:t}, e_{t+1})$.
- Since all the e_j are constant, we apply Bayes's rule and then normalize. I.e., find $\mathbf{P}(e_{t+1} | X_{t+1}, e_{1:t}) \mathbf{P}(X_{t+1} | e_{1:t})$.
- Looking more closely at the first term, we don't need to write the $e_{1:t}$ part: given X_{t+1} , e_{t+1} is independent with all e 's before.
- Now we just need normalizing $\mathbf{P}(e_{t+1} | X_{t+1}) \mathbf{P}(X_{t+1} | e_{1:t})$.
- How? Easy! The first term is just the sensor model, and the second term is just a one-step prediction from $\mathbf{f}_{1:t} = \mathbf{P}(X_t | e_{1:t})$.

AI(0270)9.50

Example: Filtering

- Let's repeat the previous example: assuming we somehow knows $P(r_5 | u_{1:5})$ is 0.6. We observed that u_6 is false.
- Recall that the one-step prediction gives $P(r_6 | u_{1:5})$ is 0.4.
- Now we must multiply with the probabilities in the CPT of the sensor model.
 - $P(r_6 | u_{1:5}) P(-u_6 | r_6) = 0.4 \times 0.1 = 0.04$.
 - $P(-r_6 | u_{1:5}) P(-u_6 | -r_6) = 0.6 \times 0.7 = 0.42$.
- So it is 2 to 21. $P(r_6 | u_{1:6}) = 0.0870$.

AI(0270)9.51

Smoothing

Once we get more information in the future, we might **go back** and find our previous beliefs are not entirely truth. This is the objective of smoothing.

- We want $\mathbf{P}(X_k | e_{1:t})$, where k is smaller than t . We already know $\mathbf{f}_{1:k} = \mathbf{P}(X_k | e_{1:k})$. So we separate 2 sets of evidences: before and after k .
- $\mathbf{P}(X_k | e_{1:t})$ is just $\mathbf{P}(X_k | e_{1:k}, e_{k+1:t})$. The e 's are all fixed, so again we applying Bayes's rule and normalize.
- So we find $\mathbf{P}(X_k | e_{1:k}) \mathbf{P}(e_{k+1:t} | X_k, e_{1:k})$ for normalization. By conditional independence, we just need $\mathbf{f}_{1:k} \mathbf{P}(e_{k+1:t} | X_k)$.
- Let's denote $\mathbf{b}_{k:t} = \mathbf{P}(e_{k:t} | X_{k-1})$, i.e., the probability of finding all the future evidences up to now, given each possible state at step $k-1$ ("backward messages").
- Then what we are normalizing is just $\mathbf{f}_{1:k} \mathbf{b}_{k+1:t}$.

AI(0270)9.52

Backward messages

- How to efficiently find $\mathbf{b}_{k:t} = \mathbf{P}(e_{k:t} | X_{k-1})$? We don't want to separately evaluate each possible sequence of X 's.
- The idea: now the probability is "given" a particular combination of X_{k-1} . If we can move it forward to X_k , we get a nice recursion just like $\mathbf{f}_{1:k}$.
- So we write $\mathbf{P}(e_{k:t} | X_{k-1}) = \sum_{x_k} \mathbf{P}(e_{k:t} | x_k, X_{k-1}) \mathbf{P}(X_k | X_{k-1})$. Due to conditional independence, this becomes $\sum_{x_k} \mathbf{P}(e_{k:t} | x_k) \mathbf{P}(X_k | X_{k-1})$.
- The term in the right is the transition model. The term in the left is not yet a recursion, though: the suffix of $e_{k:t}$ is one too small.
- So we separate out e_k , and get $\sum_{x_k} \mathbf{P}(e_k, e_{k+1:t} | x_k) \mathbf{P}(X_k | X_{k-1})$, which is $\sum_{x_k} \mathbf{P}(e_{k+1:t} | x_k) \mathbf{P}(e_k | e_{k+1:t}, x_k) \mathbf{P}(X_k | X_{k-1})$. Conditional independence simplify this to $\sum_{x_k} \mathbf{P}(e_{k+1:t} | x_k) \mathbf{P}(e_k | x_k) \mathbf{P}(X_k | X_{k-1})$.

AI(0270)9.53

Backward messages (cont'd)

Now it is clear that the recursion works:

- The $P(e_{k+1:t} | x_k)$ comes from the recursion.
- The $P(e_k | x_k)$ is just the sensor model.
- The $P(x_k | X_{k-1})$ is just the transition model.

So it can be repeated until we get $P(e_{t+1:t} | X_t)$. At this point we stop, returning just a set of 1's as probability (since the $e_{t+1:t}$ is an empty event).

The whole algorithm runs in time **linear** to the size of all the CPTs, although each of the steps is actually quite complex.

If X contains many variables, each step involves considering all **combinations** of such variables.

Special cases

If the Markov chain is more simple, more efficient algorithm can be used. Two interesting special cases:

- **Hidden Markov Model (HMM)**: If each of the state variable and the evidence variable contain only **one** variable of **discrete** values, we can use simple **matrix** operations to deal with all the sums.

Many speech recognition systems take advantage of this.

- **Kalman filter**: If the variables are **continuous** variables, the problem normally becomes too hard to solve.

But if the variables are **normally distributed** (have "Gaussian distribution"), one can operate on the parameters of the distribution. Many motion estimation systems make use of Kalman filter.

If interested, look at the textbook.