Belief network inference

Four main approaches to determine posterior distributions in belief networks:

- **Variable Elimination:** exploit the structure of the network to eliminate (sum out) the non-observed, non-query variables one at a time.
- **Stochastic simulation:** random cases are generated according to the probability distributions.
- **Search-based:** enumerate some of the possible worlds, and estimate posterior probabilities from the worlds generated.
- **Variational methods:** find the closest tractable distribution to the (posterior) distribution we are interested in. (We won't cover this one)

Factors

- A **factor** is a representation of a function from a tuple of random variables (conjunction of valued variables) into a number (a probability).
- We will write factor f on variables $X_1, \ldots, X_j$ as $f(X_1, \ldots, X_j)$.
- We can assign some or all of the variables of a factor:
  - $f(X_1 = v_1, X_2, \ldots, X_j)$, where $v_1 \in dom(X_1)$, is a factor on $X_2, \ldots, X_j$.
  - $f(X_1 = v_1, X_2 = v_2, \ldots, X_j = v_j)$ is a number that is the value of f when each X_i has value v_i.
- The former is also written as $f(X_1, X_2, \ldots, X_j)_{X_1 = v_1}$, etc.

Example factors

1. Factor on X,Y,Z is $r(X, Y, Z)$:

X	Y	Z	val
t	t	t	0.1
t	t	f	0.9
t	f	t	0.2
t	f	f	0.8
f	t	t	0.4
f	t	f	0.6
f	f	t	0.3
f	f	f	0.7

2. Factor on Y,Z is $r(X{=}t, Y, Z)$:

Y	Z	val
t	t	0.1
t	f	0.9
f	t	0.2
f	f	0.8

3. Factor on Y is $r(X{=}t, Y, Z{=}f)$:

Y	val
t	0.9
f	0.8

4. $r(X{=}t, Y{=}f, Z{=}f) = \; 0.8$

Multiplying factors

The **product** of factor $f_1(\overline{X}, \overline{Y})$ and $f_2(\overline{Y}, \overline{Z})$, where $\overline{Y}$ are the variables in common, is the factor $(f_1 \times f_2)(\overline{X}, \overline{Y}, \overline{Z})$ defined by:

$$(f_1 \times f_2)(\overline{X}, \overline{Y}, \overline{Z}) \;=\; f_1(\overline{X}, \overline{Y}) f_2(\overline{Y}, \overline{Z}).$$

Multiplying factors example

f_1:

A	B	val
t	t	0.1
t	f	0.9
f	t	0.2
f	f	0.8

f_2:

B	C	val
t	t	0.3
t	f	0.7
f	t	0.6
f	f	0.4

$f_1 \times f_2$:

A	B	C	val
t	t	t	0.03
t	t	f	0.07
t	f	t	0.54
t	f	f	0.36
f	t	t	0.06
f	t	f	0.14
f	f	t	0.48
f	f	f	0.32

Summing out variables

We can **sum out** a variable, say X_1 with domain $\{v_1, \ldots, v_k\}$, from factor $f(X_1, \ldots, X_j)$, resulting in a factor on $X_2, \ldots, X_j$ defined by:

$$
\begin{aligned}
&(\sum_{X_1} f)(X_2, \ldots, X_j) \\
&\quad = \quad f(X_1 = v_1, \ldots, X_j) + \cdots + f(X_1 = v_k, \ldots, X_j)
\end{aligned}
$$

Summing out a variable example

f_3:

A	B	C	val
t	t	t	0.03
t	t	f	0.07
t	f	t	0.54
t	f	f	0.36
f	t	t	0.06
f	t	f	0.14
f	f	t	0.48
f	f	f	0.32

$\sum_B f_3$:

A	C	val
t	t	0.57
t	f	0.43
f	t	0.54
f	f	0.46

Factors an probabilities

A conditional probability distribution $P(X|Y_1, ..., Y_j)$ can be seen as a factor f on $X, Y1, ..., Yj$, where:

$$f(X = u, Y_1 = v_1, ..., Y_j = v_j) = P(X = u | Y_1 = v_1 \wedge \cdots \wedge Y_j = v_j)$$

Evidence

If we want to compute the posterior probability of query Z given evidence $Y_1 = v_1 \wedge \ldots \wedge Y_j = v_j$:

$$
\begin{aligned}
& P(Z | Y_1 = v_1, \ldots, Y_j = v_j) \\
= \;\; & \frac{P(Z, Y_1 = v_1, \ldots, Y_j = v_j)}{P(Y_1 = v_1, \ldots, Y_j = v_j)} \\
= \;\; & \frac{P(Z, Y_1 = v_1, \ldots, Y_j = v_j)}{\sum_Z P(Z, Y_1 = v_1, \ldots, Y_j = v_j).}
\end{aligned}
$$

So the computation reduces to the probability of $P(Z, Y_1 = v_1, \ldots, Y_j = v_j)$.
We normalize at the end.

Probability of a conjunction

- Suppose the variables of the belief network are $X_1, \ldots, X_n$, with observed $\{Y_1, \ldots, Y_j\}$, and query Z
- To compute $P(Z, Y_1 = v_1, \ldots, Y_j = v_j)$
  - we sum out the other variables where $Z_1, \ldots, Z_k = \{X_1, \ldots, X_n\} - \{Z\} - \{Y_1, \ldots, Y_j\}$
  - We order the Z_i into an **elimination ordering.**

$$
\begin{aligned}
&P(Z, Y_1 = v_1, \ldots, Y_j = v_j) \\
&= \sum_{Z_k} \cdots \sum_{Z_1} P(X_1, \ldots, X_n)_{Y_1 = v_1, \ldots, Y_j = v_j}. \\
&= \sum_{Z_k} \cdots \sum_{Z_1} \prod_{i=1}^{n} P(X_i | parents(X_i))_{Y_1 = v_1, \ldots, Y_j = v_j}.
\end{aligned}
$$

by using:
$P(X_1, ..., X_n) = P(X_1 | parents(X_1)) \ldots P(X_n | parents(X_n))$

Computing sums of products

...The VE algorithm thus selects the worlds with the observed values for the Y_i's and sums over the possible worlds with the same value for Z.

Computation in belief networks reduces to computing the sums of products:

- How can we compute $ab + ac$ efficiently?
- Distribute out the a giving $a(b + c)$
- How can we compute $\sum_{Z_1} \prod_{i=1}^{n} P(X_i|parents(X_i))$ efficiently?
- Distribute out those factors that don't involve Z_1.

Variable elimination algorithm

To compute $P(Z|Y_1 = v_1 \wedge \ldots \wedge Y_j = v_j)$:

- Construct a factor for each conditional probability.
- Set the observed variables to their observed values.
- Sum out each of the other variables (the $\{Z_1, \ldots, Z_k\}$) according to some elimination ordering.
- Multiply the remaining factors. Normalize by dividing the resulting factor $f(Z)$ by $\sum_Z f(Z)$.

Summing out a variable

To sum out a variable Z_j from a product $f_1, \ldots, f_k$ of factors:

- Partition the factors into
  - those that don't contain Z_j, say $f_1, \ldots, f_i$,
  - those that contain Z_j, say $f_{i+1}, \ldots, f_k$

We know:

$$\sum_{Z_j} f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_i \times \left(\sum_{Z_j} f_{i+1} \times \cdots \times f_k \right).$$

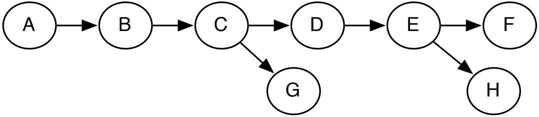
- VE explicitly constructs a representation (in terms of a multidimensional array, a tree, or a set of rules) of the rightmost factor. Replace the factors $f_{i+1}, \ldots, f_k$ by the new factor.

VE algorithm

```
Procedure VE_BN(Vs,Ps,O,Q)
    Inputs
        Vs: set of variables
        Ps: set of factors representing the conditional probabilities
        O: set of observations of values on some of the variables
        Q: a query variable
    Output
        posterior distribution on Q
    Local
        Fs: a set of factors
    Fs ←Ps
    for each X∈Vs-{Q} using some elimination ordering do
        if (X is observed) then
            for each F ∈Fs that involves X do
                set X in F to its observed value in O
                project F onto remaining variables
        else
            Rs←{F ∈Fs: F  involves X}
            let T be the product of the factors in Rs
            N ←∑_X T
            Fs←Fs \ Rs∪{N}
    let T be the product of the factors in Fs
    N ←∑_Q T
    return T/N
```

Inference
Variable elimination
Factors
VE algorithm
Improving VE
Stochastic methods
Samples
Cumulative dist.
Forward sampling
Rejection sampling
Importance sampling
Particle filtering
Temporal models
Markov chains
HMM
Filtering
Smoothing
Other methods
HMM-particle
DBN
Search

Variable Elimination example



Query: $P(G|f)$; elimination ordering: A, H, E, D, B, C

$$P(G|f) \propto \sum_C \sum_B \sum_D \sum_E \sum_H \sum_A P(A)P(B|A)P(C|B) P(D|C)P(E|D)P(f|E)P(G|C)P(H|E)$$

$$= \sum_C \left(\sum_B \left(\sum_A P(A)P(B|A) \right) P(C|B) \right) P(G|C) \left(\sum_D P(D|C) \left(\sum_E P(E|D)P(f|E) \sum_H P(H|E) \right) \right)$$

For efficiency: prune irrelevant variables

Suppose you want to compute $P(X|e_1 \ldots e_k)$:

- Prune any variables that have no observed or queried descendents.
- Connect the parents of any observed variable.
- Remove arc directions.
- Remove observed variables.
- Remove any variables not connected to X in the resulting (undirected) graph.

To improve efficiency of VE

Heuristics:

- **min-factor:** at each stage, select the variable that results in the smallest relation or size of the next factor
- **minimum deficiency or minimum fill:** select variable that adds smallest number of arcs to remaining constraint network. Deficiency of a variable X is the number of pairs of variables that are in a relationship with X that are not in a relationship with each other. Okay to remove a variable that results in a large relation as long as it does not make the network more complicated.

Or, abandon exact inference all together, and go with stochastic methods...

Stochastic methods

- Set of samples can be used to compute probabilities.
- E.g., probability P(a)=0.14 means that, out of 1,000 samples, about 140 will have a true.
- You can go from (enough) samples into probabilities and from probabilities into samples.
- We consider three problems:
  1. how to generate samples,
  2. how to incorporate observations, and
  3. how to infer probabilities from samples.
- Three methods:
  1. rejection sampling,
  2. importance sampling, and
  3. particle filtering.

Stochastic and Monte Carlo methods

Monte Carlo methods are a broad class of algorithms that rely on repeated random sampling to obtain results about an problem intractable to compute exactly.

- **Idea:** probabilities $\leftrightarrow$ samples
- Get probabilities from samples:

X	$count$
x_1	n_1
$\vdots$	$\vdots$
x_k	n_k
$total$	m

$\leftrightarrow$

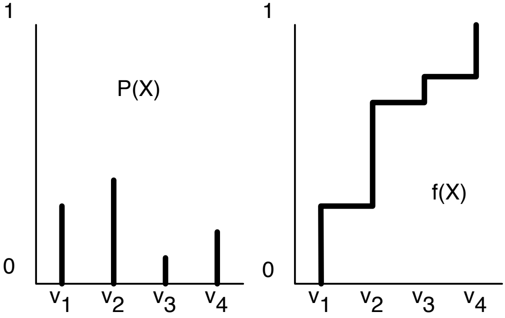
X	$probability$
x_1	n_1/m
$\vdots$	$\vdots$
x_k	n_k/m

- If we could sample from a variable's (posterior) probability, we could estimate its (posterior) probability.

Generating samples from a distribution

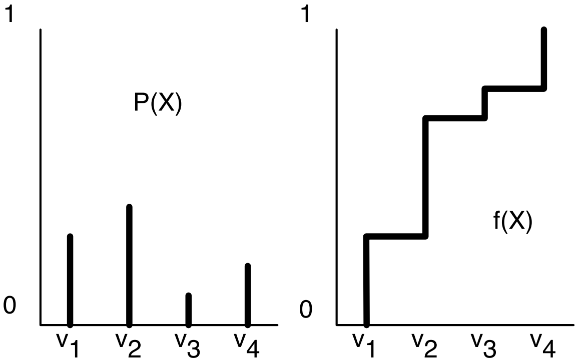
For a variable X with a discrete domain or a (one-dimensional) real domain:

- Totally order the values of the domain of X (obvious and trivial usually), left plot.
- Generate the cumulative probability distribution at right: $f(x) = P(X \leq x)$.



Generating samples from a distribution

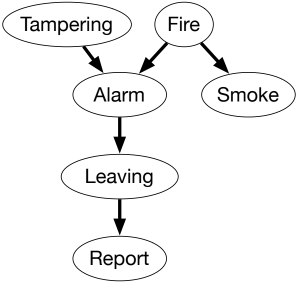
- Select a value y **uniformly** in the range $[0, 1]$.
- Select the x such that $f(x) = y$.



Forward sampling in a belief network

- Total ordering of the variables so that the parents of a variable come before the variable in the total order
- Sample the variables one at a time; sample parents of X before sampling X.
- Given values for the parents of X, sample from the probability of X given its parents.

Forward sampling in a belief network



Sample	Tampering	Fire	Alarm	Smoke	Leaving	Report
s1	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
s2	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
s3	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
s4	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
s5	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
s6	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
s7	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE
s8	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE
...						
s1000	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE

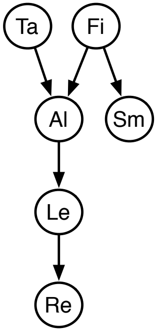
Rejection Sampling

- Given some evidence e, rejection sampling estimates: $P(h|e) = (P(h \wedge e))/(P(e))$
- Consider only the samples where e is true and by determining the proportion of these in which h is true.
- Samples are generated as in forward, but any sample where e is false is rejected immediately.
- Proportion of the remaining, non-rejected, samples where h is true is an estimate of $P(h|e)$
- The non-rejected samples are distributed according to the posterior probability:

$$P(\alpha|evidence) \approx \frac{\sum_{sample \models \alpha} 1(sample)}{\sum_{sample} 1(sample)}$$

  where we consider only samples consistent with evidence.

Rejection Sampling Example: $P(ta|sm, re)$



Observe $Sm = true, Re = true$

	Ta	Fi	Al	Sm	Le	Re	
s_1	false	true	false	true	false	false	✘
s_2	false	true	true	true	true	true	✔
s_3	true	false	true	false	—	—	✘
s_4	true	true	true	true	true	true	✔
...							
s_{1000}	false	false	false	false	—	—	✘

$P(sm) = 0.02$
$P(re|sm) = 0.32$
How many samples are rejected?
How many samples are used?

Problem: Rejection sampling does not work well when the evidence is unlikely.

Importance Sampling

Solution: Don't sample uniformly based on priors, but then adjust accordingly.

- Samples have weights: a real number associated with each sample that takes the evidence into account.
- Probability of a proposition is weighted average of samples:

$$P(\alpha|evidence) \approx \frac{\sum_{sample \models \alpha} weight(sample)}{\sum_{sample} weight(sample)}$$

- Mix exact inference with sampling: don't sample all of the variables, but weight each sample according to $P(evidence|sample)$.
- A.K.A Likelihood weighting

Importance sampling

Importance sampling differs from rejection sampling:

1. Importance sampling does not sample all variables, only some of them. The variables that are not sampled and are not observed are summed out. Don't sample the observed variables (although the algorithm does not preclude this).
2. Importance sampling does not have to sample the variables according to their prior probability. The distribution that it uses to sample the variables is called the **proposal distribution**, q.

Conceptual example

Suppose $P(a) = 0.98$: $A = \mathit{false}$ would only be true in about 20 samples out of 1,000.

Instead, $A = \mathit{true}$ is sampled 50% of the time, but each sample with A=true can be weighted by $0.98/0.5 = 1.96$ and
each sample with $A = \mathit{false}$ can be weighted by $0.02/0.5 = 0.04$.

Example proposal distribution

We want to compute: $P(alarm|smoke \wedge report)$

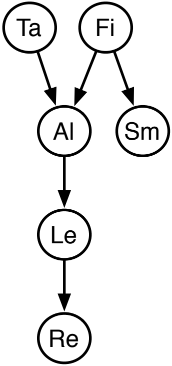
Proposal distribution:

- $q(tampering) = 0.02$
- $q(fire) = 0.5$
- $q(Alarm|Tampering, Fire) = P(Alarm|Tampering, Fire)$
- $q(Leaving|Alarm) = P(Leaving|Alarm)$

Particulars:

- $p = P(e|s)P(s)/q(s)$ is the weight of the sample.
- e is $smoke \wedge report$
- $P(e|s)$ is equal to $P(smoke|Fire)P(report|Leaving)$
- $P(s)/q(s)$ is 0.02 when $Fire = true$ in the sample and is 1.98 when $Fire = false$;

Importance Sampling Example:



$P(alarm|smoke \wedge report)$

Ta	Fi	Al	Le	$P(e\|s)$	$P(s)/q(s)$	p
FALSE	TRUE	FALSE	TRUE	0.675	0.02	0.0135
TRUE	TRUE	TRUE	FALSE	0.009	0.02	0.00018
FALSE	FALSE	FALSE	TRUE	0.0075	1.98	0.01485
FALSE	TRUE	FALSE	FALSE	0.009	0.02	0.00018

$$P(h|e) = lim_{n\to\infty} 1/k \sum_{s_i} (P(h|s_i, e)P(e|s_i)P(s_i))/q(s_i)$$

Particle filtering

- Importance sampling enumerates the samples one at a time and, for each sample, assigns a value to each variable.
- The particle filtering algorithm generates all the samples for one variable before moving to the next variable.

Particle filtering

Steps:

- Select a variable that has not been sampled or summed out and is not observed. For each particle, sample the variable according to some proposal distribution. Weight of the particle is updated as in importance sampling.
- Weight of particle is multiplied by probability of evidence given values of particle
- Resample the population. Resampling constructs a new population of particles, each with the same weight, by selecting particles from the population, where each particle is chosen with probability proportional to the weight of the particle. Some particles may be forgotten and some may be duplicated.

Particle filtering

Benefits:

1. it can be used for an unbounded number of variables (which we will see later).
2. the particles better cover the hypothesis space. Whereas importance sampling will involve some particles that have very low probability, with only a few of the particles covering most of the probability mass, resampling lets many particles more uniformly cover the probability mass.

Temporal models

We can model a dynamic system as a belief network by treating a feature at a particular time as a random variable. We first give a model in terms of states and then show how it can be extended to features.

These will be important later for reinforcement learning.

Markov chain

- A **Markov chain** is a special sort of belief network:

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4$$

What probabilities need to be specified? What Independence assumptions are made?

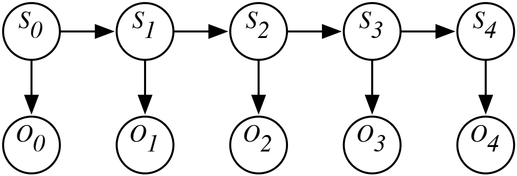
- Often S_t represents the **state** at time t.
- $P(S_0)$ specifies initial conditions
- $P(S_{t+1}|S_t)$ specifies the dynamics
- $P(S_{t+1}|S_0, \ldots, S_t) = P(S_{t+1}|S_t)$ which is called the **Markov assumption**.
  - Intuitively S_t conveys all of the information about the history that can affect the future states.
  - "The future is independent of the past given the present."

Stationary Markov chain

- A **stationary Markov chain** is when for all $t > 0$, $t' > 0$, $P(S_{t+1}|S_t) = P(S_{t'+1}|S_{t'})$.
- We specify $P(S_0)$ and $P(S_{t+1}|S_t)$.
- It is of interest because:
  - Simple model, easy to specify
  - Often the natural model
  - The network can extend indefinitely
  - To determine the probability distribution of state S_i, VE can be used to sum out the preceding variables. Note that the variables after S_i are irrelevant to the probability of S_i and need not be considered.

Hidden Markov Model

- A **Hidden Markov Model (HMM)** is a belief network:



The probabilities that need to be specified:

- $P(S_0)$ specifies initial conditions
- $P(S_{t+1}|S_t)$ specifies the dynamics
- $P(O_t|S_t)$ specifies the sensor model

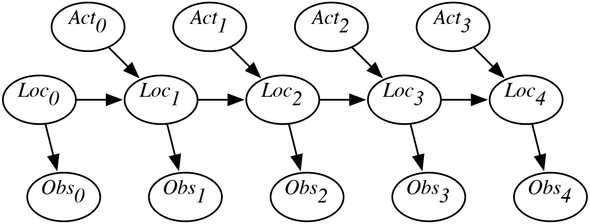
Filtering

What is the current belief state based on the observation history?

- Compute the probability of the current state given the history of observations.
- For each i, the agent wants to compute $P(S_i|o_1, \ldots, o_i)$, which is the distribution over the state at time i given the particular observation of $o_0, \ldots, o_i$.
- This can easily be done using VE $P(S_i|o_1, \ldots, o_i) = P(o_i|S_i) \sum_{S_{i-1}} P(S_i|S_{i-1}) P(S_{i-1}|o_0, \ldots, o_{i-1})$

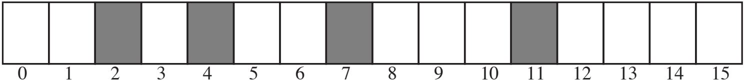
Example: localization

- Suppose a robot wants to determine its location based on its actions and its sensor readings: **Localization**
- This can be represented by the augmented HMM:



Example localization domain

- Circular corridor, with 16 locations:



- Doors at positions: 2, 4, 7, 11.
- Noisy Sensors
- Stochastic Dynamics
- Robot starts at an unknown location and must determine where it is.

Example Sensor Model

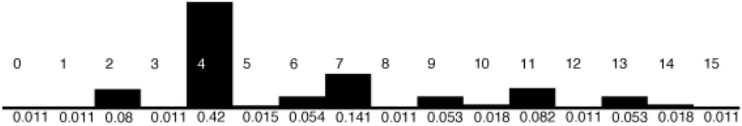
- $P(Observe\ Door \mid At\ Door) = 0.8$
- $P(Observe\ Door \mid Not\ At\ Door) = 0.1$

In 20% of the cases in which the robot is at a door, the sensor falsely gives a negative reading. In 10% of the cases where the robot is not at a door, the sensor records that there is a door.

Example Dynamics Model

- $P(loc_{t+1} = L | action_t = goRight \wedge loc_t = L) = 0.1$
- $P(loc_{t+1} = L+1 | action_t = goRight \wedge loc_t = L) = 0.8$
- $P(loc_{t+1} = L+2 | action_t = goRight \wedge loc_t = L) = 0.074$
- $P(loc_{t+1} = L' | action_t = goRight \wedge loc_t = L) = 0.002$ for any other location L'.

Example Dynamics Model



- Goal: Robot starts at an unknown location and must determine its location.
- Robot's probability distribution over its locations, assuming it starts with no knowledge of where it is and experiences the following observations: **(observe door, go right, observe no door, go right, and then observe door)**.
- Location 4 is the most likely current location, with posterior probability of 0.42.

Smoothing

- Smoothing is the problem of computing the probability distribution of a state variable in an HMM given past and future observations.
- The use of future observations can make for more accurate predictions.
- Suppose an agent has observed up to time k and wants to determine the state at time i for $i < k$; the smoothing problem is to determine

  $$P(S_i | o_1, \ldots, o_k)$$

  All of the variables for $i > k$ can be ignored.
- Given a new observation it is possible to update all previous state estimates with one sweep through the states using VE

Extending temporal models

In addition to monitoring and smoothing, the following can be performed:

- **Prediction:** Compute posterior distribution over future states. Naturally, this decreases in quality as time extends
- **Most likely explanation:** Given observations, which states were most likely to have generated the observations
- **Learning:** Transition and sensor models themselves can be learned from observations

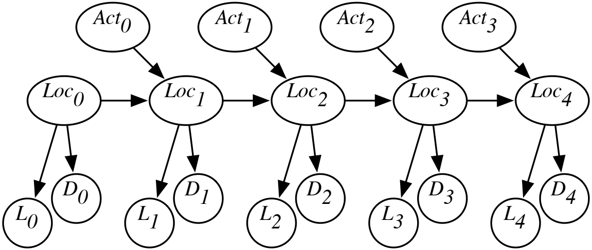
Particle Filtering for HMMs

- Start with a number of random chosen particles (say 1000)
- Each particle represents a state, selected in proportion to the initial probability of the state.
- Repeat:
  - Absorb evidence: weight each particle by the probability of the evidence given the state represented by the particle.
  - Resample: select each particle at random, in proportion to the weight of the sample.
    Some particles may be duplicated, some may be removed.
  - Transition: sample the next state for each particle according to the transition probabilities.

To answer a query about the current state, use the set of particles as data.

Combining sensor information

- **Example:** we can combine information from a light sensor and the door sensor **Sensor Fusion**



S_t robot location at time t
D_t door sensor value at time t
L_t light sensor value at time t

Combining state information

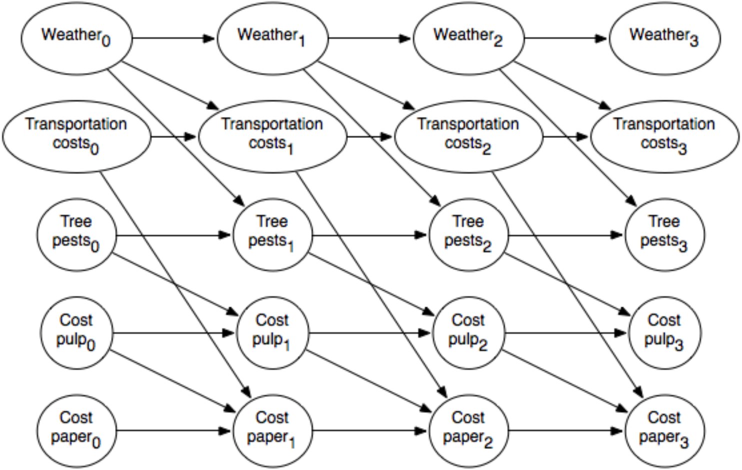
If multiple states are required to be tracked, the single-state model of an HMM can be fed a tuple of each state, as an aggregate single state.

Problem: this is inefficient, like tabulating the entire joint distribution, rather than using the Bayesian network.

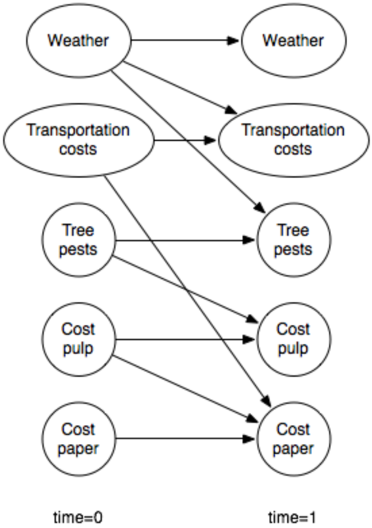
Dynamic belief/Bayesian networks

- **Solution:** A dynamic belief network (DBN) is like a Hiddden Markov model, but the states and the observations are represented in terms of features.
- If F is a feature, we write F_t as the random variable that represented the value of variable F at time t.
- A dynamic belief network makes the following assumptions:
  - The set of features is the same at each time.
  - For any time $t > 0$, the parents of variable F_t are variables at time t or time $t-1$, such that the graph for any time is acyclic.
  - The conditional probability distribution of how each variable depends on its parents is the same for every time $t > 0$.

Dynamic belief networks



Dynamic belief networks



Searching Possible Worlds (not required to know)

- Can we estimate the probabilities by only enumerating a few of the possible worlds?
- How can we enumerate just a few of the most probable possible worlds?
- Can we estimate the error in our estimates?
- Can we exploit the structure that variable elimination does?
- Can we exploit more structure?

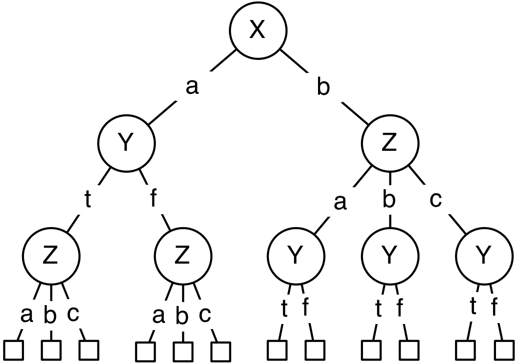
Search tree

The **search tree** has nodes labeled with variables, and is defined as follows:

- Each non-leaf node is labelled with a variable
- The arcs are labelled with values. There is a child for a node X for every value in the domain of X.
- A node cannot be labelled with the same label as an ancestor node.
- A path from the root corresponds to an assignment to a set of variables.
- In a full tree, every path from the root to a leaf contains all variables. The leaves correspond to possible worlds.

Example search tree

Suppose we have 3 variables, X with domain $\{a, b\}$, Y with domain $\{t, f\}$, and Z with domain $\{a, b, c\}$:



Basic Search Algorithm

$Q := \{\langle\rangle\};$
$W := \{\};$
While $Q \neq \{\}$ do
  choose and remove $\langle Y_1{=}v_1, \cdots, Y_j{=}v_j\rangle$ from Q;
  if $j = n$
    $W \leftarrow W \cup \{\langle Y_1{=}v_1, \cdots, Y_j{=}v_j\rangle\}$
  else
    Select a variable $Y_{j+1} \notin \{Y_1, \ldots, Y_n\}$
    $Q \leftarrow Q \cup \{\langle Y_1{=}v_1, \cdots, Y_j{=}v_j, Y_{j+1}{=}v\rangle : v \in dom(Y_{j+1})\}$

Q is a set of paths from root to a leaf.
W is a set of generated possible worlds.

Properties of the Algorithm

- Each partial description can only be generated once. There is no need to check for multiple paths or loops in the search.
- The probability of a world W is

$$\prod_i P(X_i | parents(X_i))_W$$

- Once a factor is fully assigned, we can multiply by its value.

Estimating the Probabilities

Use W, at the start of an iteration of the while loop, as an approximation to the set of all possible worlds.
Let

$$P_W^g = \sum_{w \in W \wedge w \models g} P(w)$$

$$P_Q = 1 - P_W^{true}$$

Then

$$P_W^g \leq P(g) \leq P_W^g + P_Q$$

Posterior Probabilities

Given the definition of conditional probability:

$$P(g|obs) = \frac{P(g \wedge obs)}{P(obs)}$$

We estimate the probability of a conditional probability:

$$\frac{P_W^{g \wedge obs}}{P_W^{obs} + P_Q} \leq P(g|obs) \leq \frac{P_W^{g \wedge obs} + P_Q}{P_W^{obs} + P_Q}$$

If we choose the midpoint as an estimate:

$$\text{Error} \quad \leq \quad \frac{P_Q}{2(P_W^{obs} + P_Q)}$$

As the computation progresses, the probability mass in the queue P_Q approaches zero.

Refinements

- We only need to consider the ancestors of the variables we are interested in. We can prune the rest before the search.
- When computing $P(\alpha)$, we prune partial descriptions if it can be determined whether α is true or false in that partial description.
- When computing $P(\bullet|OBS)$, we prune partial descriptions in which OBS is false.
- We want to generate the most likely possible worlds to minimize the error. One good search strategy is a depth-first search, pruning unlikely worlds.

Recursive Conditioning

- Consider a factor graph where the nodes are factors and there are arcs between two factors that have a variables in common.
- Assigning a value v to a variable X, simplifies all factors that contain X.
  Factor F that contains X becomes factor $F_{X=v}$ which doesn't contain X.
- If an assignment disconnects the graph, each component can be evaluated separately.
- Computed values can be cached. The cache can be checked before evaluating any query.

Recursive Conditioning

procedure $rc(Fs:$ set of factors$)$:
  if $Fs = \{\}$ return 1
  else if $\exists v$ such that $\langle Fs, v\rangle \in cache$
    return v
  else if $\exists F \in Fs$ such that $vars(F) = \{\}$
    return $F \times rc(Fs \setminus F)$
  else if $Fs = Fs_1 \uplus Fs_2$ such that $vars(Fs_1) \cap vars(Fs_2) = \{\}$
    return $rc(Fs_1) \times rc(Fs_2)$
  else select variable $X \in vars(Fs)$
    $sum \leftarrow 0$
    for each $v \in dom(X)$
      $sum \leftarrow sum + rc(\{F_{X=v} : F \in Fs\})$
    $cache \leftarrow cache \cup \{\langle Fs, sum\rangle\}$
    return sum

Notes on the $rc(Fs)$ algorithm

- *cache* is a global variable that contains sets of pairs. It is initially empty.
- $vars(F)$ returns the unassigned variables in F
- $F_{X=v}$ is F with variable X assigned to value v
- $Fs = Fs_1 \uplus Fs_2$ is the disjoint union, meaning $Fs_1 \neq \{\}$, $Fs_2 \neq \{\}$, $Fs_1 \cap Fs_2 = \{\}$, $Fs = Fs_1 \cup Fs_2$ This step recognizes when the graph is disconnected.

Exploiting Structure in Recursive Conditioning

- How can we exploit determinism (zero probabilities)?
- How can we exploit context-specific independencies. E.g., if $P(X|Y=y, Z=z) = P(X|Y=y, Z=z')$ for a particular y and for all values z, z'?