Lecture 10

Inference in Baysian Networks and Reasoning Over Time

Marco Chiarandini

Deptartment of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- ✔ Adversarial Search
  - ✔ Minimax search
  - ✔ Alpha-beta pruning
- ✔ Knowledge representation and Reasoning
  - ✔ Propositional logic
  - ✔ First order logic
  - ✔ Inference
- Uncertain knowledge and Reasoning
  - Probability and Bayesian approach
  - Bayesian Networks
  - Hidden Markov Chains
  - Kalman Filters
- Learning
  - Decision Trees
  - Maximum Likelihood
  - EM Algorithm
  - Learning Bayesian Networks
  - Neural Networks
  - Support vector machines

Outline

1. Inference by Randomized Algorithms
2. Exercise
3. Uncertainty over Time

Complexity of exact inference

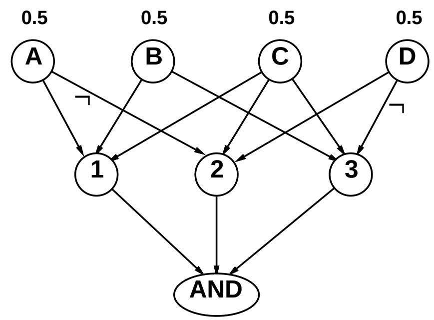
Singly connected networks (or polytrees):
– any two nodes are connected by at most one (undirected) path
– time and space cost (with variable elimination) are $O(d^k n)$
– hence time and space cost are linear in n and k bounded by a constant

Multiply connected networks:
– can reduce 3SAT to exact inference $\implies$ NP-hard
– equivalent to **counting** 3SAT models $\implies$ #P-complete

1. A v B v C

2. C v D v ¬A

3. B v C v ¬D

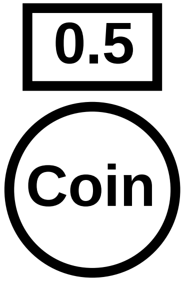


Inference by stochastic simulation

Basic idea:

- Draw N samples from a sampling distribution S
- Compute an approximate posterior probability $\hat{P}$
- Show this converges to the true probability P

Outline:
– Sampling from an empty network
– Rejection sampling: reject samples disagreeing with evidence
– Likelihood weighting: use evidence to weight samples
– Markov chain Monte Carlo (MCMC): sample from a stochastic process
whose stationary distribution is the true posterior



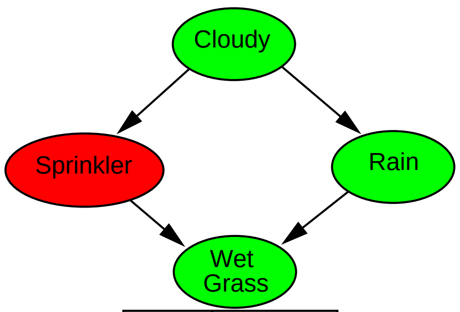
Sampling from an empty network

function Prior-Sample(bn) **returns** an event sampled from bn
inputs: bn, a belief network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

$\mathbf{x} \leftarrow$ an event with n elements
for $i = 1$ **to** n **do**
$x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
given the values of $Parents(X_i)$ in $\mathbf{x}$
return $\mathbf{x}$

Example

P(C)
.50



C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Sampling from an empty network contd.

Probability that PriorSample generates a particular event

$$S_{PS}(x_1 \ldots x_n) = P(x_1 \ldots x_n)$$

i.e., the true prior probability

E.g., $S_{PS}(t, f, t, t) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324 = P(t, f, t, t)$

Proof: Let $N_{PS}(x_1 \ldots x_n)$ be the number of samples generated for event $x_1, \ldots, x_n$. Then we have

$$\begin{aligned}
\lim_{N\to\infty} \hat{P}(x_1, \ldots, x_n) &= \lim_{N\to\infty} N_{PS}(x_1, \ldots, x_n)/N \\
&= S_{PS}(x_1, \ldots, x_n) \\
&= \prod_{i=1}^{n} P(x_i | parents(X_i)) = P(x_1 \ldots x_n)
\end{aligned}$$

$\leadsto$That is, estimates derived from PriorSample are consistent
Shorthand: $\hat{P}(x_1, \ldots, x_n) \approx P(x_1 \ldots x_n)$

Rejection sampling

$\hat{\mathbf{P}}(X|\mathbf{e})$ estimated from samples agreeing with $\mathbf{e}$

```
function Rejection-Sampling(X, e, bn, N) returns an estimate of P(X|e)
   local variables: N, a vector of counts over X, initially zero

   for j = 1 to N do
      x ← Prior-Sample(bn)
      if x is consistent with e then
         N[x] ← N[x]+1 where x is the value of X in x
   return Normalize(N[X])
```

E.g., estimate $\mathbf{P}(Rain|Sprinkler = true)$ using 100 samples
27 samples have $Sprinkler = true$
Of these, 8 have $Rain = true$ and 19 have $Rain = false$.

$\hat{\mathbf{P}}(Rain|Sprinkler = true) = \text{Normalize}(\langle 8, 19\rangle) = \langle 0.296, 0.704\rangle$
Similar to a basic real-world empirical estimation procedure

Analysis of rejection sampling

Rejection sampling returns consistent posterior estimates

Proof:
$\hat{\mathbf{P}}(X|\mathbf{e}) = \alpha \mathbf{N}_{PS}(X, \mathbf{e})$ (algorithm defn.)
$= \mathbf{N}_{PS}(X, \mathbf{e})/N_{PS}(\mathbf{e})$ (normalized by $N_{PS}(\mathbf{e})$)
$\approx \mathbf{P}(X, \mathbf{e})/P(\mathbf{e})$ (property of PriorSample)
$= \mathbf{P}(X|\mathbf{e})$ (defn. of conditional probability)

Problem: hopelessly expensive if $P(\mathbf{e})$ is small
$P(\mathbf{e})$ drops off exponentially with number of evidence variables!

Likelihood weighting

Idea: fix evidence variables, sample only nonevidence variables,
and weight each sample by the likelihood it accords the evidence

```
function Likelihood-Weighting(X, e, bn, N) returns an estimate of P(X|e)
   local variables: W, a vector of weighted counts over X, initially zero

   for j = 1 to N do
      x, w ← Weighted-Sample(bn)
      W[x] ← W[x] + w where x is the value of X in x
   return Normalize(W[X])
```

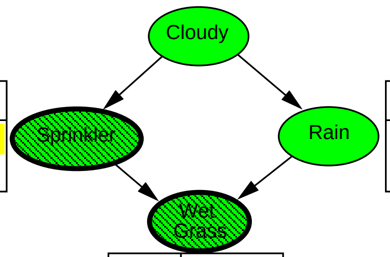
```
function Weighted-Sample(bn, e) returns an event and a weight

   x ← an event with n elements; w ← 1
   for i = 1 to n do
      if X_i has a value x_i in e
         then w ← w × P(X_i = x_i | parents(X_i))
         else x_i ← a random sample from P(X_i | parents(X_i))
   return x, w
```


Likelihood weighting example

$P(Rain|Sprinkler = true, WetGrass = true)$

P(C)
.50



C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

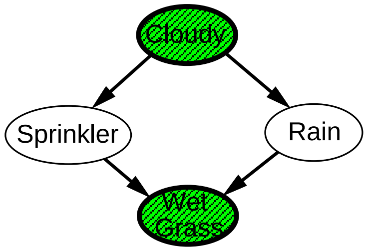
S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Likelihood weighting analysis

Likelihood weighting returns consistent estimates

Sampling probability for WeightedSample is

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | parents(Z_i))$$



(pays attention to evidence in **ancestors** only)
⇝somewhere "in between" prior and posterior distribution

Weight for a given sample $\mathbf{z}, \mathbf{e}$ is

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | parents(E_i))$$

but performance still degrades with many evidence variables because a few samples have nearly all the total weight

Weighted sampling probability is

$$S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | parents(Z_i)) \prod_{i=1}^{m} P(e_i | parents(E_i)) = P(\mathbf{z}, \mathbf{e})$$

Summary

Approximate inference by LW:
- LW does poorly when there is lots of (late-in-the-order) evidence
- LW generally insensitive to topology
- Convergence can be very slow with probabilities close to 1 or 0
- Can handle arbitrary combinations of discrete and continuous variables

Approximate inference using MCMC

"State" of network = current assignment to all variables.
Generate next state by sampling one variable given Markov blanket
Sample each variable in turn, keeping evidence fixed

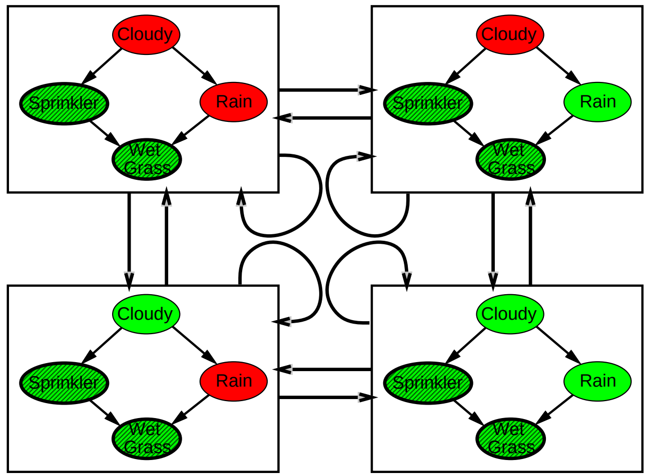
function MCMC-Ask(X, $\mathbf{e}$, bn, N) **returns** an estimate of $P(X|\mathbf{e})$
  local variables: $\mathbf{N}[X]$, a vector of counts over X, initially zero
    $\mathbf{Z}$, nonevidence variables in bn, hidden + query
    $\mathbf{x}$, current state of the network, initially copied from $\mathbf{e}$

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

Can also choose a variable to sample at random each time

The Markov chain

With $Sprinkler = true, WetGrass = true$, there are four states:



Wander about for a while, average what you see

Probabilistic finite state machine

MCMC example contd.

Estimate $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$

Sample *Cloudy* or *Rain* given its Markov blanket, repeat.
Count number of times *Rain* is true and false in the samples.

E.g., visit 100 states
31 have $Rain = true$, 69 have $Rain = false$

$\hat{\mathbf{P}}(Rain|Sprinkler = true, WetGrass = true) = \text{Normalize}(\langle 31, 69\rangle) = \langle 0.31, 0.69\rangle$

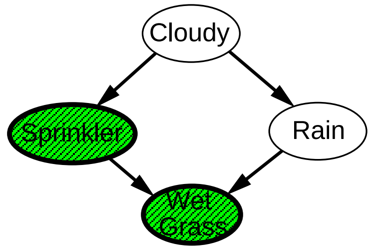
Theorem

The Markov Chain approaches a stationary distribution:
long-run fraction of time spent in each state is exactly proportional to its posterior probability

Markov blanket sampling

Markov blanket of *Cloudy* is
Sprinkler and *Rain*

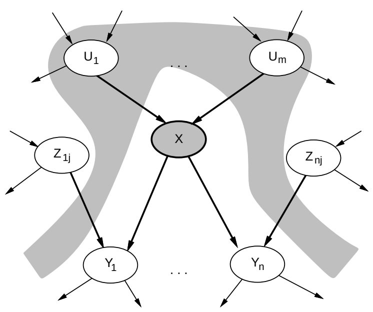
Markov blanket of *Rain* is
Cloudy, *Sprinkler*, and *WetGrass*



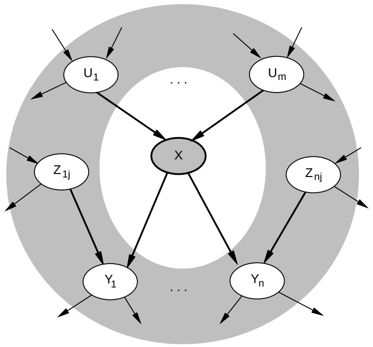
Main computational problems:
1) Difficult to tell if convergence has been achieved
2) Can be wasteful if Markov blanket is large:
$P(X_i|mb(X_i))$ won't change much (law of large numbers)

Local semantics and Markov Blanket

Local semantics: each node is conditionally independent of its nondescendants given its parents



Each node is conditionally independent of all others given its Markov blanket: parents + children + children's parents



MCMC analysis: Outline

- Transition probability $q(\mathbf{x} \to \mathbf{x}')$
- Occupancy probability $\pi_t(\mathbf{x})$ at time t
- Equilibrium condition on π_t defines stationary distribution $\pi(\mathbf{x})$
  Note: stationary distribution depends on choice of $q(\mathbf{x} \to \mathbf{x}')$
- Pairwise detailed balance on states guarantees equilibrium
- Gibbs sampling transition probability:
  sample each variable given current values of all others
  $\Longrightarrow$ detailed balance with the true posterior
- For Bayesian networks, Gibbs sampling reduces to sampling conditioned on each variable's Markov blanket

Stationary distribution

- $\pi_t(\mathbf{x})$ = probability in state $\mathbf{x}$ at time t
  $\pi_{t+1}(\mathbf{x}')$ = probability in state $\mathbf{x}'$ at time $t+1$
- π_{t+1} in terms of π_t and $q(\mathbf{x} \to \mathbf{x}')$

$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \to \mathbf{x}')$$

- Stationary distribution: $\pi_t = \pi_{t+1} = \pi$

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \to \mathbf{x}') \qquad \text{for all } \mathbf{x}'$$

- If π exists, it is unique (specific to $q(\mathbf{x} \to \mathbf{x}')$)
- In equilibrium, expected "outflow" = expected "inflow"

Detailed balance

- "Outflow" = "inflow" for each pair of states:

$$\pi(\mathbf{x}) q(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \to \mathbf{x}) \qquad \text{for all } \mathbf{x},\ \mathbf{x}'$$

- Detailed balance $\implies$ stationarity:

$$\begin{aligned} \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \to \mathbf{x}') &= \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \to \mathbf{x}) \\ &= \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \to \mathbf{x}) \\ &= \pi(\mathbf{x}') \end{aligned}$$

- MCMC algorithms typically constructed by designing a transition probability q that is in detailed balance with desired π

Gibbs sampling

- Sample each variable in turn, given **all other variables**
- Sampling X_i, let $\bar{\mathbf{X}}_i$ be all other nonevidence variables
- Current values are x_i and $\bar{\mathbf{x}}_i$; $\mathbf{e}$ is fixed
- Transition probability is given by

$$q(\mathbf{x} \to \mathbf{x}') = q(x_i, \bar{\mathbf{x}}_i \to x_i', \bar{\mathbf{x}}_i) = P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e})$$

- This gives detailed balance with true posterior $P(\mathbf{x}|\mathbf{e})$:

$$\begin{aligned} \pi(\mathbf{x}) q(\mathbf{x} \to \mathbf{x}') &= P(\mathbf{x}|\mathbf{e}) P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) = P(x_i, \bar{\mathbf{x}}_i | \mathbf{e}) P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) \\ &= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e}) P(\bar{\mathbf{x}}_i | \mathbf{e}) P(x_i' | \bar{\mathbf{x}}_i, \mathbf{e}) \quad \text{(chain rule)} \\ &= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e}) P(x_i', \bar{\mathbf{x}}_i | \mathbf{e}) \qquad \text{(chain rule backwards)} \\ &= q(\mathbf{x}' \to \mathbf{x}) \pi(\mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \to \mathbf{x}) \end{aligned}$$

Performance of approximation algorithms

- Absolute approximation: $|P(X|\mathbf{e}) - \hat{P}(X|\mathbf{e})| \leq \epsilon$
- Relative approximation: $\frac{|P(X|\mathbf{e}) - \hat{P}(X|\mathbf{e})|}{P(X|\mathbf{e})} \leq \epsilon$
- Relative $\implies$ absolute since $0 \leq P \leq 1$ (may be $O(2^{-n})$)
- Randomized algorithms may fail with probability at most δ
- Polytime approximation: $\mathrm{poly}(n, \epsilon^{-1}, \log \delta^{-1})$
- Theorem (Dagum and Luby, 1993): both absolute and relative approximation for either deterministic or randomized algorithms are NP-hard for any $\epsilon, \delta < 0.5$
  (Absolute approximation polytime with no evidence—Chernoff bounds)

Summary

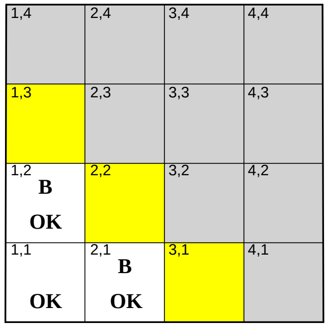
Exact inference by variable elimination:
- polytime on polytrees, NP-hard on general graphs
- space = time, very sensitive to topology

Approximate inference by LW, MCMC:

– PriorSampling and RejectionSampling unusable as evidence grow
- LW does poorly when there is lots of (late-in-the-order) evidence
- LW, MCMC generally insensitive to topology
- Convergence can be very slow with probabilities close to 1 or 0
- Can handle arbitrary combinations of discrete and continuous variables

Outline

Wumpus World



$P_{ij} = true$ iff $[i, j]$ contains a pit
$B_{ij} = true$ iff $[i, j]$ is breezy
Include only $B_{1,1}, B_{1,2}, B_{2,1}$ in the probability model

Specifying the probability model

The full joint distribution is $\mathbf{P}(P_{1,1}, \ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$

Apply product rule: $\mathbf{P}(B_{1,1}, B_{1,2}, B_{2,1} \mid P_{1,1}, \ldots, P_{4,4})\mathbf{P}(P_{1,1}, \ldots, P_{4,4})$

(Do it this way to get $P(Effect|Cause)$.)

First term: 1 if pits are adjacent to breezes, 0 otherwise

Second term: pits are placed randomly, probability 0.2 per square:

$$\mathbf{P}(P_{1,1}, \ldots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathbf{P}(P_{i,j}) = 0.2^n \times 0.8^{16-n}$$

for n pits.

Observations and query

We know the following facts:

$b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$
$known = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$

Query is $\mathbf{P}(P_{1,3}|known, b)$

Define $Unknown$ = P_{ij}s other than $P_{1,3}$ and $Known$

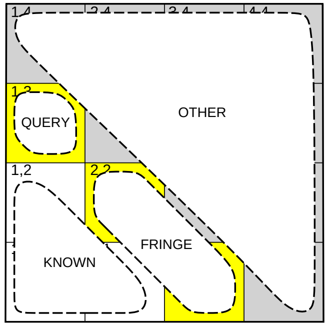
For inference by enumeration, we have

$$\mathbf{P}(P_{1,3}|known, b) = \alpha \sum_{unknown} \mathbf{P}(P_{1,3}, unknown, known, b)$$

Grows exponentially with number of squares!

Using conditional independence

Basic insight: observations are conditionally independent of other hidden squares given neighbouring hidden squares

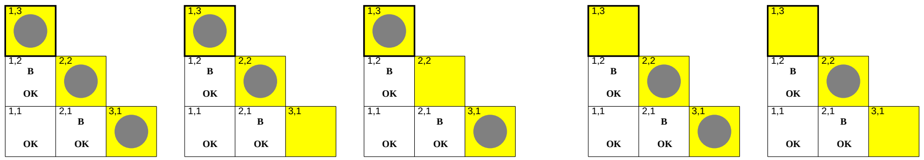


Define $Unknown = Fringe \cup Other$
$\mathbf{P}(b|P_{1,3}, Known, Unknown) = \mathbf{P}(b|P_{1,3}, Known, Fringe)$
Manipulate query into a form where we can use this!

Using conditional independence contd.

$$\begin{aligned}
\mathbf{P}(P_{1,3}|known, b) &= \alpha \sum_{unknown} \mathbf{P}(P_{1,3}, unknown, known, b) \\
&= \alpha \sum_{unknown} \mathbf{P}(b|P_{1,3}, known, unknown)\mathbf{P}(P_{1,3}, known, unknown) \\
&= \alpha \sum_{fringe} \sum_{other} \mathbf{P}(b|known, P_{1,3}, fringe, other)\mathbf{P}(P_{1,3}, known, fringe, other) \\
&= \alpha \sum_{fringe} \sum_{other} \mathbf{P}(b|known, P_{1,3}, fringe)\mathbf{P}(P_{1,3}, known, fringe, other) \\
&= \alpha \sum_{fringe} \mathbf{P}(b|known, P_{1,3}, fringe) \sum_{other} \mathbf{P}(P_{1,3}, known, fringe, other) \\
&= \alpha \sum_{fringe} \mathbf{P}(b|known, P_{1,3}, fringe) \sum_{other} \mathbf{P}(P_{1,3})P(known)P(fringe)P(other) \\
&= \alpha\, P(known)\mathbf{P}(P_{1,3}) \sum_{fringe} \mathbf{P}(b|known, P_{1,3}, fringe)P(fringe) \sum_{other} P(other) \\
&= \alpha'\, \mathbf{P}(P_{1,3}) \sum_{fringe} \mathbf{P}(b|known, P_{1,3}, fringe)P(fringe)
\end{aligned}$$

Using conditional independence contd.



0.2 x 0.2 = 0.04   0.2 x 0.8 = 0.16   0.8 x 0.2 = 0.16   0.2 x 0.2 = 0.04   0.2 x 0.8 = 0.16

$$\begin{aligned}
\mathbf{P}(P_{1,3}|known, b) &= \alpha' \langle 0.2(0.04 + 0.16 + 0.16),\ 0.8(0.04 + 0.16)\rangle \\
&\approx \langle 0.31, 0.69\rangle
\end{aligned}$$

$$\mathbf{P}(P_{2,2}|known, b) \approx \langle 0.86, 0.14\rangle$$

Outline

Outline

- ◇ Time and uncertainty
- ◇ Inference: filtering, prediction, smoothing
- ◇ Hidden Markov models
- ◇ Kalman filters (a brief mention)
- ◇ Dynamic Bayesian networks (an even briefer mention)

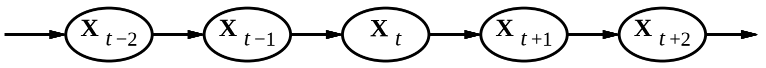
Time and uncertainty

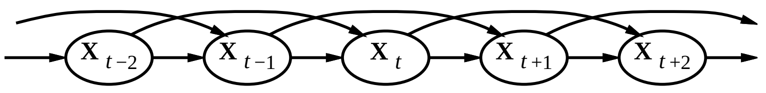
- The world changes; we need to track and predict it
- Diabetes management vs vehicle diagnosis
- Basic idea: copy state and evidence variables for each time step
  $\mathbf{X}_t$ = set of unobservable state variables at time t
  e.g., $BloodSugar_t$, $StomachContents_t$, etc.
  $\mathbf{E}_t$ = set of observable evidence variables at time t
  e.g., $MeasuredBloodSugar_t$, $PulseRate_t$, $FoodEaten_t$
- This assumes **discrete time**; step size depends on problem
- Notation: $\mathbf{X}_{a:b} = \mathbf{X}_a, \mathbf{X}_{a+1}, \ldots, \mathbf{X}_{b-1}, \mathbf{X}_b$

Markov processes (Markov chains)

Construct a Bayes net from these variables:
- unbounded number of conditional probability table
- unbounded number of parents

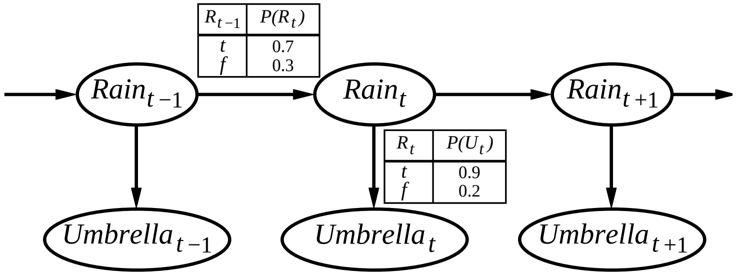
Markov assumption: $\mathbf{X}_t$ depends on **bounded** subset of $\mathbf{X}_{0:t-1}$
First-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$
Second-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$

First−order


Second−order


Sensor Markov assumption: $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$
$\rightsquigarrow$Stationary process:
- transition model $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ and
- sensor model $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ fixed for all t

Example



First-order Markov assumption not exactly true in real world!
Possible fixes:

1. **Increase order** of Markov process
2. **Augment state**, e.g., add $Temp_t$, $Pressure_t$

Example: robot motion.
Augment position and velocity with $Battery_t$

Inference tasks

1. Filtering: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$
   belief state—input to the decision process of a rational agent

2. Prediction: $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for $k > 0$
   evaluation of possible action sequences;
   like filtering without the evidence

3. Smoothing: $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ for $0 \leq k < t$
   better estimate of past states, essential for learning

4. Most likely explanation: $\arg\max_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}|\mathbf{e}_{1:t})$
   speech recognition, decoding with a noisy channel

Filtering

Aim: devise a **recursive** state estimation algorithm:

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t}))$$

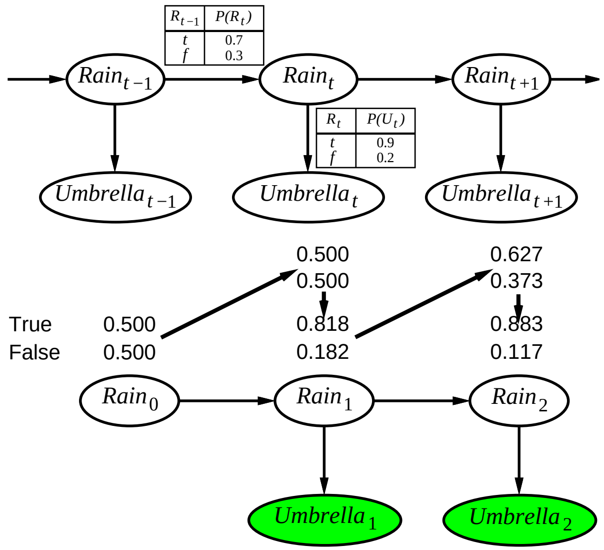
$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \\ &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) \\ &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})\end{aligned}$$

I.e., prediction + estimation. Prediction by summing out $\mathbf{X}_t$:

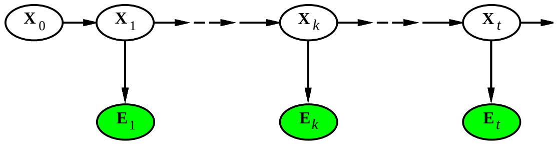
$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\sum_{\mathbf{x}_t}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t})P(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\sum_{\mathbf{x}_t}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t})\end{aligned}$$

$\mathbf{f}_{1:t+1} = \text{Forward}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1})$ where $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$
Time and space **constant** (independent of t)

Filtering example



Smoothing



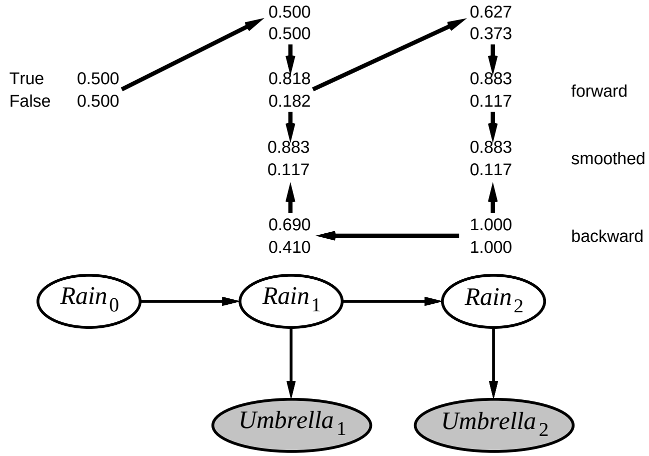
Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

$$\begin{aligned}
\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{e}_{1:k}) \\
&= \alpha\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \\
&= \alpha\mathbf{f}_{1:k}\mathbf{b}_{k+1:t}
\end{aligned}$$

Backward message computed by a backwards recursion:

$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)
\end{aligned}$$

Smoothing example



Forward–backward algorithm: cache forward messages along the way
Time linear in t (polytree inference), space $O(t|\mathbf{f}|)$