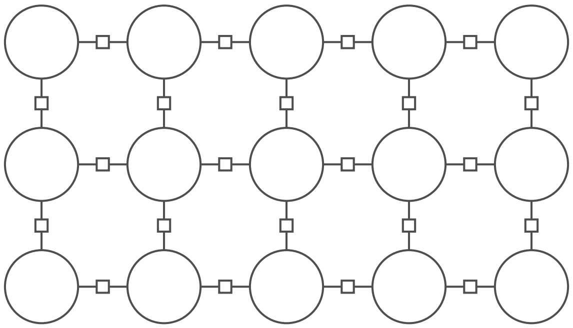
Bayesian Networks II



Lecture: Bayesian networks

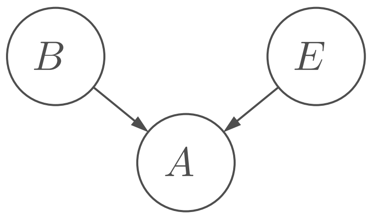
Definitions: Probabilistic Programming

Inference: Probabilistic Inference

Inference: Forward Backward

Inference: Particle Filtering

Probabilistic programs



Joint distribution:

$$\mathbb{P}(B = b, E = e, A = a) = p(b)p(e)p(a \mid b, e)$$

Probabilistic program: alarm

$B \sim \text{Bernoulli}(\epsilon)$

$E \sim \text{Bernoulli}(\epsilon)$

$A = B \vee E$

```
def Bernoulli(epsilon):
    return random.random() < epsilon
```

Key idea: probabilistic program

A randomized program that sets the random variables.

Probabilistic program: example

Probabilistic program: object tracking

$X_0 = (0, 0)$

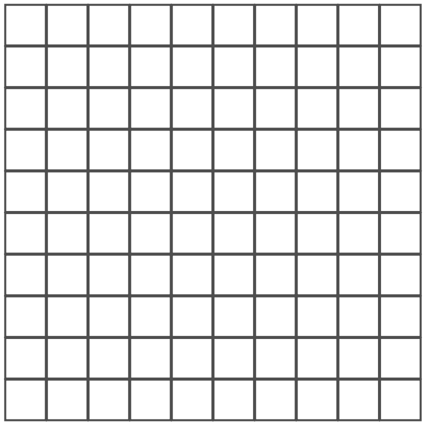
For each time step $i = 1, \dots, n$:

  if Bernoulli(α):

    $X_i = X_{i-1} + (1, 0)$ [go right]

  else:

    $X_i = X_{i-1} + (0, 1)$ [go down]



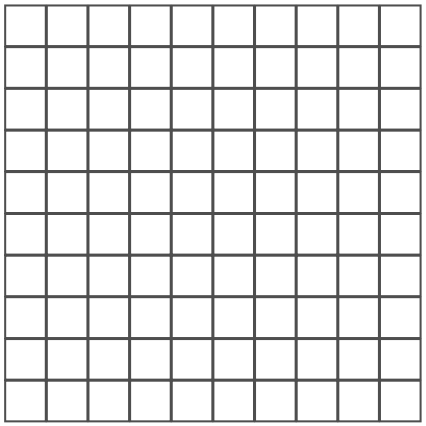
(press ctrl-enter to save)

Run

$$X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5$$

Probabilistic inference: example

Question: what are possible trajectories given **evidence** $X_{10} = (8, 2)$?



(press ctrl-enter to save)

Run

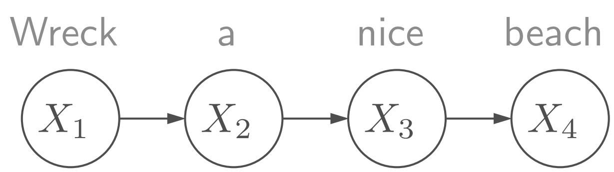
Application: language modeling

Can be used to score sentences for speech recognition or machine translation

Probabilistic program: Markov model

For each position $i = 1, 2, \ldots, n$:

Generate word $X_i \sim p(X_i \mid X_{i-1})$



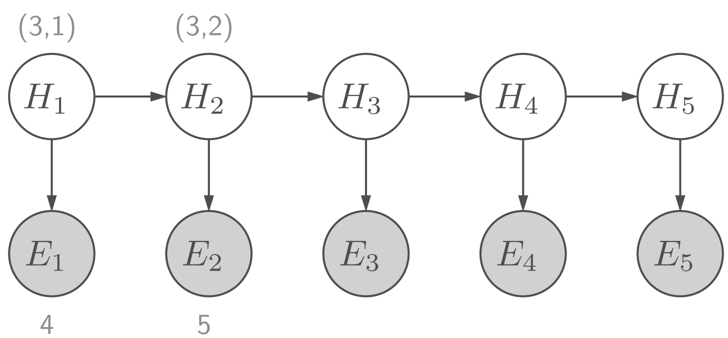
Application: object tracking

Probabilistic program: hidden Markov model (HMM)

For each time step $t = 1, \dots, T$:

Generate object location $H_t \sim p(H_t \mid H_{t-1})$

Generate sensor reading $E_t \sim p(E_t \mid H_t)$



Inference: given sensor readings, where is the object?

Application: multiple object tracking

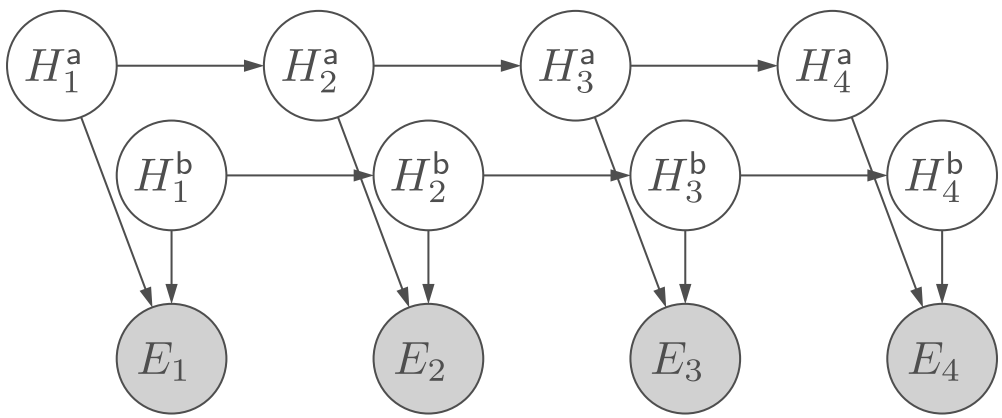
Probabilistic program: factorial HMM

For each time step $t = 1, \dots, T$:

  For each object $o \in \{\mathsf{a}, \mathsf{b}\}$:

    Generate location $H_t^o \sim p(H_t^o \mid H_{t-1}^o)$

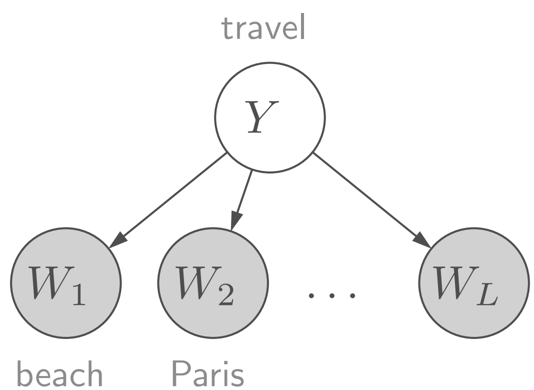
  Generate sensor reading $E_t \sim p(E_t \mid H_t^{\mathsf{a}}, H_t^{\mathsf{b}})$



Application: document classification

Probabilistic program: naive Bayes

Generate label $Y \sim p(Y)$
For each position $i = 1, \dots, L$:
    Generate word $W_i \sim p(W_i \mid Y)$



Inference: given a text document, what is it about?

Application: topic modeling

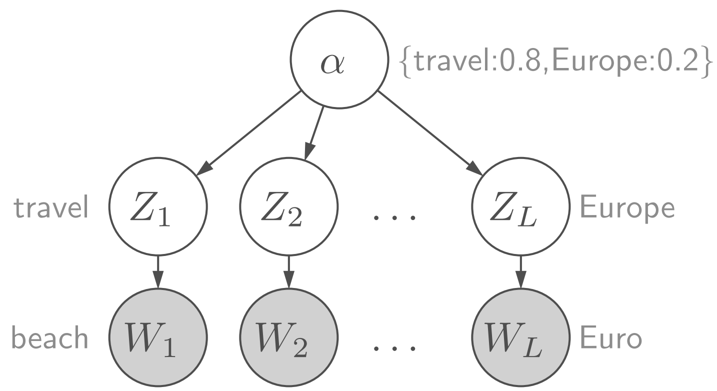
Probabilistic program: latent Dirichlet allocation

Generate a distribution over topics $\alpha \in \mathbb{R}^K$

For each position $i = 1, \dots, L$:

  Generate a topic $Z_i \sim p(Z_i \mid \alpha)$

  Generate a word $W_i \sim p(W_i \mid Z_i)$



Inference: given a text document, what topics is it about?

Application: medical diagnosis

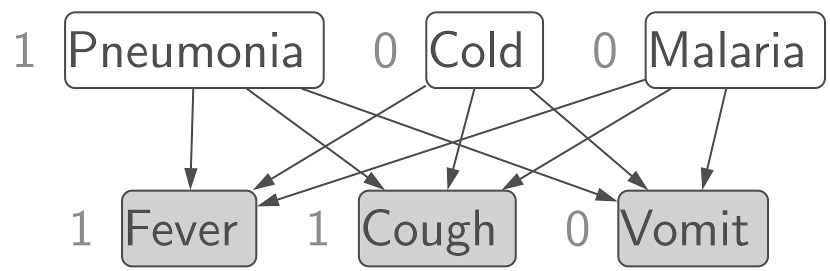
Probabilistic program: diseases and symptoms

For each disease $i = 1, \dots, m$:

  Generate activity of disease $D_i \sim p(D_i)$

For each symptom $j = 1, \dots, n$:

  Generate activity of symptom $S_j \sim p(S_j \mid D_{1:m})$



Inference: If a patient has some symptoms, what diseases do they have?

Application: social network analysis

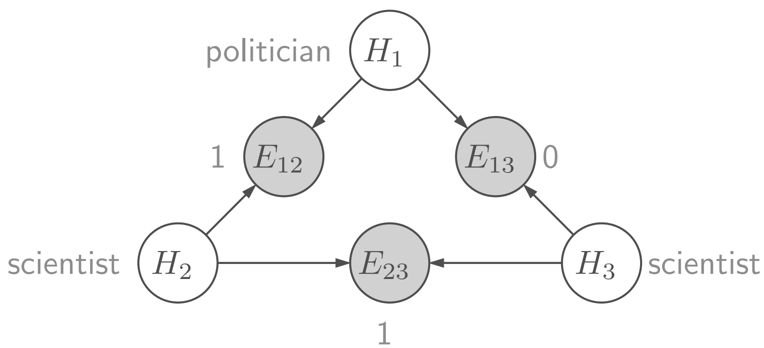
Probabilistic program: stochastic block model

For each person $i = 1, \dots, n$:

  Generate person type $H_i \sim p(H_i)$

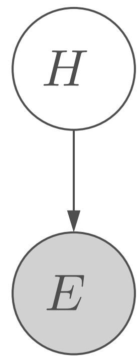
For each pair of people $i \neq j$:

  Generate connectedness $E_{ij} \sim p(E_{ij} \mid H_i, H_j)$



Inference: Given a social network graph, what types of people are there?

Summary



- Probabilistic program specifies a Bayesian network
- Many different types of models
- Common paradigm: come up with stories of how the quantities of interest (output) generate the data (input)
- Opposite of how we normally do classification!

Lecture: Bayesian networks

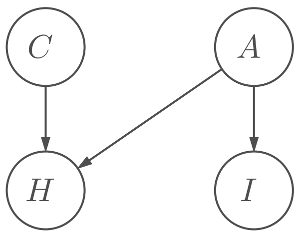
Definitions: Probabilistic Programming

Inference: Probabilistic Inference

Inference: Forward Backward

Inference: Particle Filtering

Review: Bayesian network



Random variables:

cold C, allergies A, cough H, itchy eyes I

Joint distribution:

$$\mathbb{P}(C = c, A = a, H = h, I = i) = p(c)p(a)p(h \mid c, a)p(i \mid a)$$

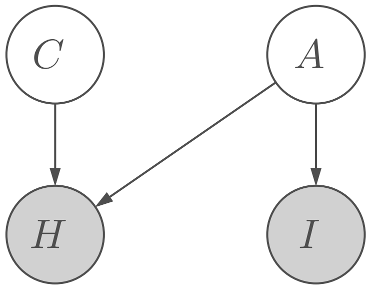
Definition: Bayesian network

Let $X = (X_1, \dots, X_n)$ be random variables.

A **Bayesian network** is a directed acyclic graph (DAG) that specifies a joint distribution over X as a product of local conditional distributions, one for each node:

$$\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) \stackrel{\text{def}}{=} \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

Review: probabilistic inference



Question: $\mathbb{P}(C \mid H = 1, I = 1)$

Input

Bayesian network: $\mathbb{P}(X_1, \dots, X_n)$

Evidence: $E = e$ where $E \subseteq X$ is subset of variables

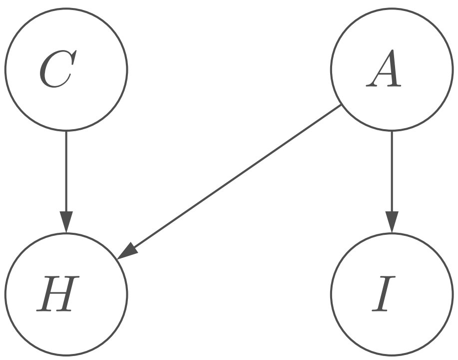
Query: $Q \subseteq X$ is subset of variables

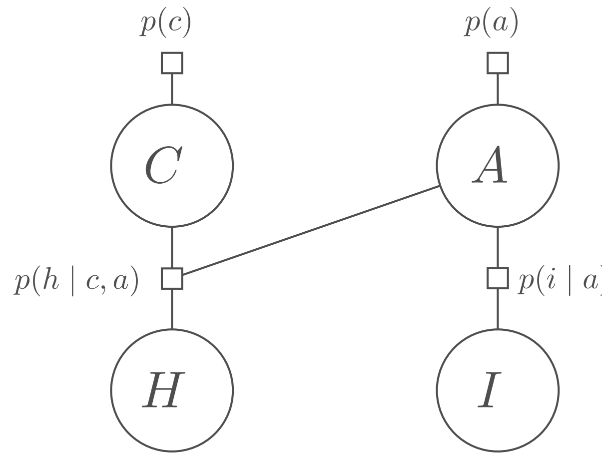
$\downarrow$

Output

$\mathbb{P}(Q \mid E = e) \longleftrightarrow \mathbb{P}(Q = q \mid E = e)$ for all values q

Reduction to Markov networks



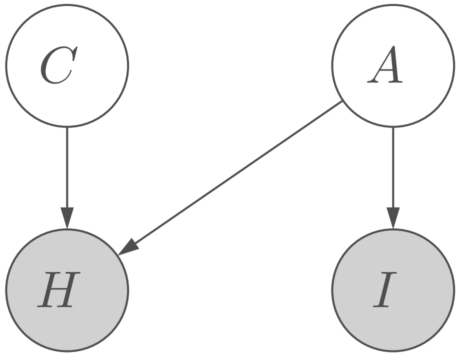


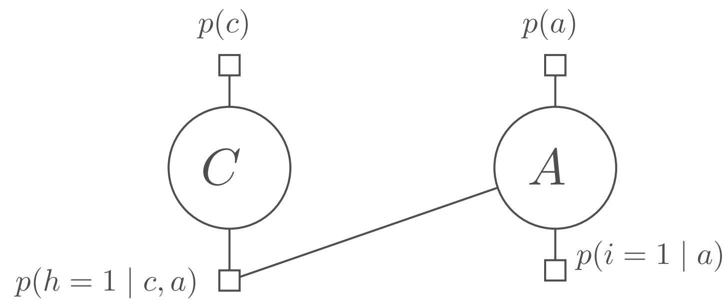
$$\mathbb{P}(C = c, A = a, H = h, I = i) = \frac{1}{Z} p(c)p(a)p(h \mid c, a)p(i \mid a)$$

> Bayesian network = Markov network with normalization constant $Z = 1$

Reminder: single factor that connects **all** parents!

Conditioning on evidence





Markov network:

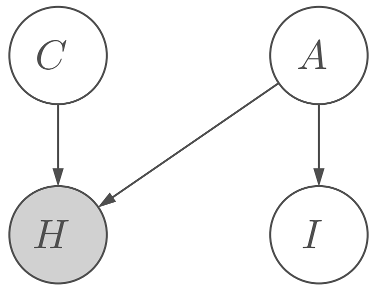
$$\mathbb{P}(C = c, A = a \mid H = 1, I = 1) = \frac{1}{Z} p(c) p(a) p(h = 1 \mid c, a) p(i = 1 \mid a)$$

Bayesian network with evidence = Markov network with $Z = \mathbb{P}(H = 1, I = 1)$

Solution: run any inference algorithm for Markov networks (e.g., Gibbs sampling)!

[demo]

Leveraging additional structure: unobserved leaves



Markov network:

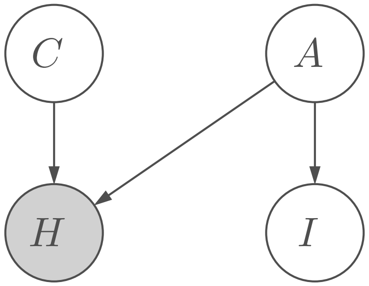
$$\mathbb{P}(C = c, A = a, I = i \mid H = 1) = \frac{1}{Z} p(c) p(a) p(h = 1 \mid c, a) p(i \mid a),$$

where $Z = \mathbb{P}(H = 1)$

Question: $\mathbb{P}(C = 1 \mid H = 1)$

Can we reduce the Markov network before running inference?

Leveraging additional structure: unobserved leaves

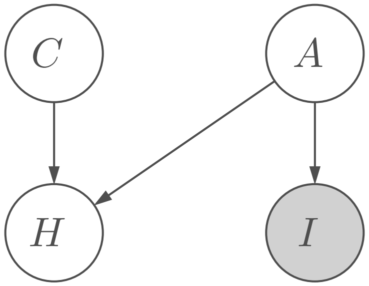


Markov network:

$$\begin{aligned}\mathbb{P}(C=c, A=a \mid H=1) &= \textstyle\sum_i \mathbb{P}(C=c, A=a, I=i \mid H=1) \\ &= \textstyle\sum_i \frac{1}{Z} p(c)p(a)p(h=1 \mid c,a)p(i \mid a) \\ &= \textstyle\frac{1}{Z} p(c)p(a)p(h=1 \mid c,a) \sum_i p(i \mid a) \\ &= \textstyle\frac{1}{Z} p(c)p(a)p(h=1 \mid c,a)\end{aligned}$$

Throw away any unobserved leaves before running inference!

Leveraging additional structure: independence

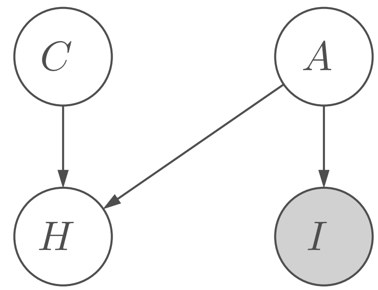


Markov network:

$$
\begin{aligned}
\mathbb{P}(C = c \mid I = 1) &= \textstyle\sum_{a,h} \mathbb{P}(C = c, A = a, H = h \mid I = 1) \\
&= \textstyle\sum_{a,h} \frac{1}{Z} p(c) p(a) p(h \mid c, a) p(i = 1 \mid a) \\
&= \textstyle\sum_{a} \frac{1}{Z} p(c) p(a) p(i = 1 \mid a) \\
&= p(c) \textstyle\sum_{a} \frac{1}{Z} p(a) p(i = 1 \mid a) \\
&= p(c)
\end{aligned}
$$

> Throw away any disconnected components before running inference!

Summary



- Condition on evidence (e.g., $I = 1$)
- Throw away unobserved leaves, e.g., I for $\mathbb{P}(C = 1 \mid H = 1)$
- Discard disconnected components, e.g., A and I for $\mathbb{P}(C = c \mid I = 1)$)
- Define Markov network out of remaining factors
- Run your favorite inference algorithm (e.g., manual, Gibbs sampling)

Lecture: Bayesian networks

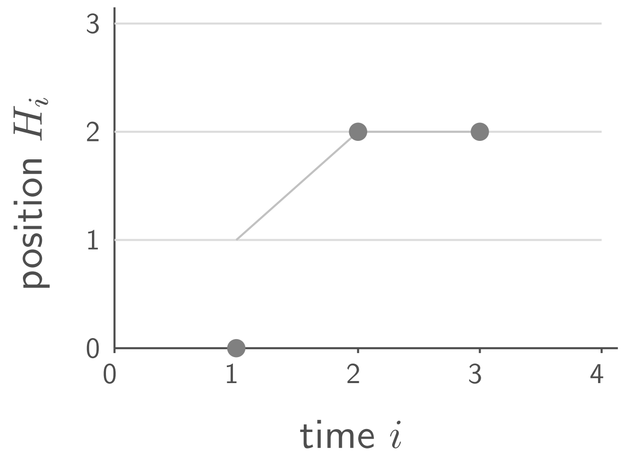
Definitions: Probabilistic Programming

Inference: Probabilistic Inference

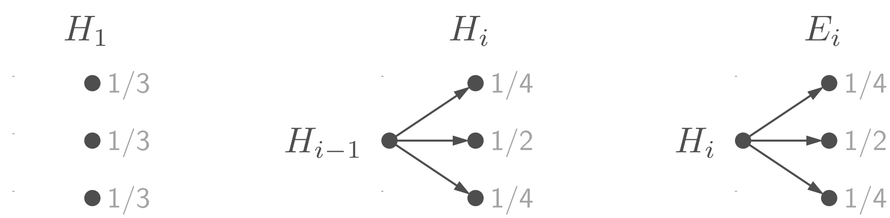
Inference: Forward Backward

Inference: Particle Filtering

Hidden Markov models for object tracking



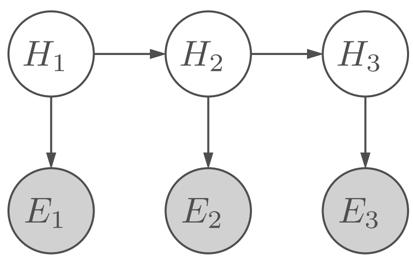
start | transition | emission



h_1	$p(h_1)$
0	1/3
1	1/3
2	1/3

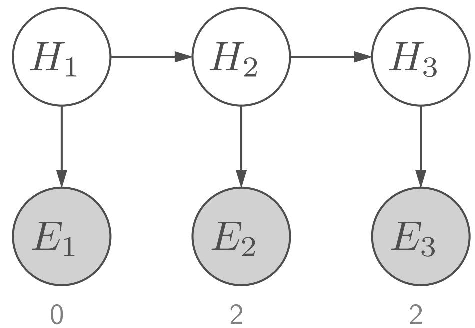
h_i	$p(h_i \mid h_{i-1})$
$h_{i-1} - 1$	1/4
h_{i-1}	1/2
$h_{i-1} + 1$	1/4

e_i	$p(e_i \mid h_i)$
$h_i - 1$	1/4
h_i	1/2
$h_i + 1$	1/4



$$\mathbb{P}(H = h, E = e) = \underbrace{p(h_1)}_{\text{start}} \prod_{i=2}^{n} \underbrace{p(h_i \mid h_{i-1})}_{\text{transition}} \prod_{i=1}^{n} \underbrace{p(e_i \mid h_i)}_{\text{emission}}$$

Inference questions



Question (**filtering**):

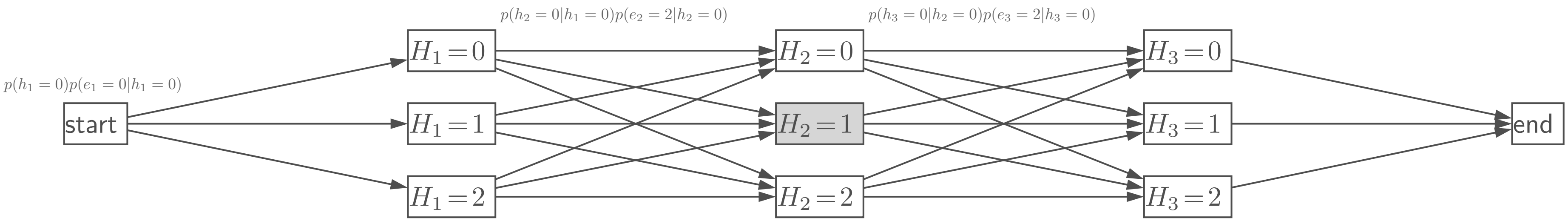
$$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2)$$

Question (**smoothing**):

$$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2, E_3 = 2)$$

Note: filtering is a special case of smoothing if marginalize unobserved leaves

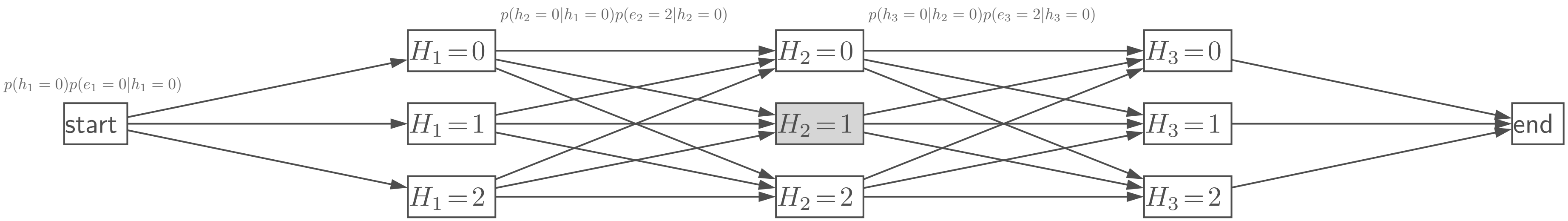
Lattice representation



- Edge $\boxed{\text{start}} \Rightarrow \boxed{H_1 = h_1}$ has weight $p(h_1)p(e_1 \mid h_1)$
- Edge $\boxed{H_{i-1} = h_{i-1}} \Rightarrow \boxed{H_i = h_i}$ has weight $p(h_i \mid h_{i-1})p(e_i \mid h_i)$
- Each path from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ is an assignment with weight equal to the product of edge weights

Key: $\mathbb{P}(H_i = h_i \mid E = e)$ is the weighted fraction of paths through $\boxed{H_i = h_i}$

Forward and backward messages



Forward: $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1})\text{Weight}(\boxed{H_{i-1} = h_{i-1}}, \boxed{H_i = h_i})$

sum of weights of paths from $\boxed{\text{start}}$ to $\boxed{H_i = h_i}$

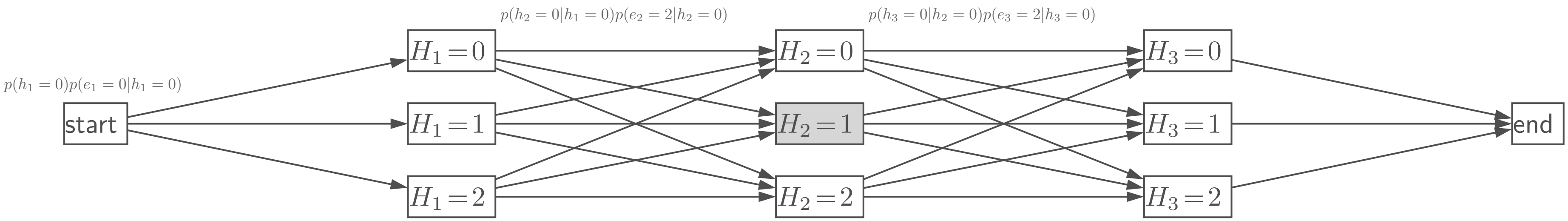
Backward: $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1})\text{Weight}(\boxed{H_i = h_i}, \boxed{H_{i+1} = h_{i+1}})$

sum of weights of paths from $\boxed{H_i = h_i}$ to $\boxed{\text{end}}$

Define $S_i(h_i) = F_i(h_i)B_i(h_i)$:

sum of weights of paths from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ through $\boxed{H_i = h_i}$

Putting everything together



$$\mathbb{P}(H_i = h_i \mid E = e) = \frac{S_i(h_i)}{\sum_v S_i(v)}$$

Algorithm: forward-backward algorithm

Compute $F_1, F_2, \dots, F_n$

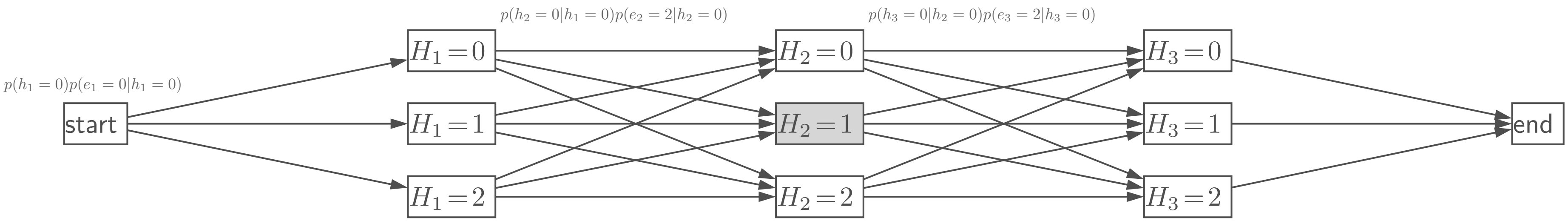
Compute $B_n, B_{n-1}, \dots, B_1$

Compute S_i for each i and normalize

Running time: $O(n|\text{Domain}|^2)$

[demo]

Summary



- Lattice representation: paths are assignments

- Dynamic programming: compute sums efficiently

- Forward-backward algorithm: compute all smoothing questions, share intermediate computations

Lecture: Bayesian networks

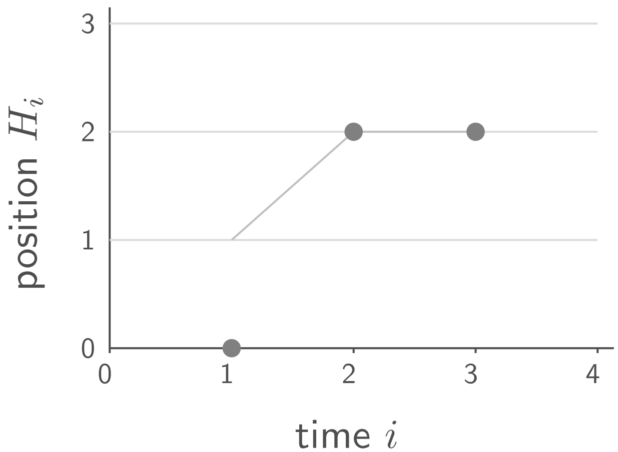
Definitions: Probabilistic Programming

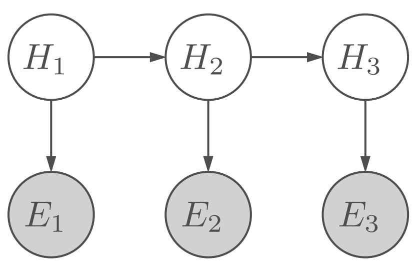
Inference: Probabilistic Inference

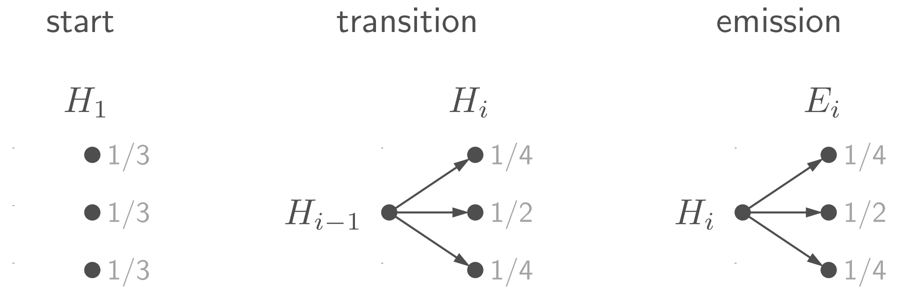
Inference: Forward Backward

Inference: Particle Filtering

Review: Hidden Markov models for object tracking







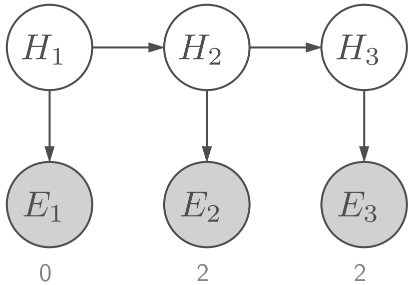
h_1	$p(h_1)$
0	1/3
1	1/3
2	1/3

h_i	$p(h_i \mid h_{i-1})$
$h_{i-1} - 1$	1/4
h_{i-1}	1/2
$h_{i-1} + 1$	1/4

e_i	$p(e_i \mid h_i)$
$h_i - 1$	1/4
h_i	1/2
$h_i + 1$	1/4

$$\mathbb{P}(H = h, E = e) = \underbrace{p(h_1)}_{\text{start}} \prod_{i=2}^{n} \underbrace{p(h_i \mid h_{i-1})}_{\text{transition}} \prod_{i=1}^{n} \underbrace{p(e_i \mid h_i)}_{\text{emission}}$$

Review: inference in Hidden Markov models



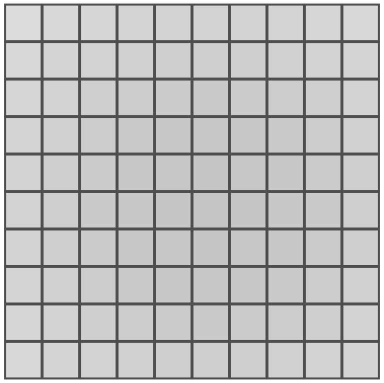
Filtering questions:

$\mathbb{P}(H_1 \mid E_1 = 0)$

$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2)$

$\mathbb{P}(H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

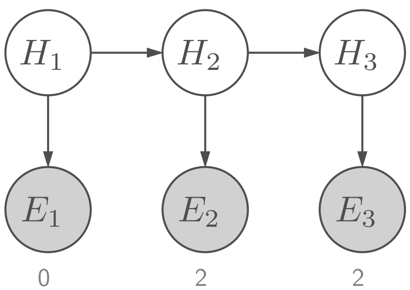
Problem: many possible location values for H_i



Forward-backward is too slow ($O(n|\text{Domain}|^2)$)...

Beam search for HMMs

Idea: keep $\leq K$ partial assignments (**particles**)



Algorithm: beam search

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \dots, n$:

Extend:

$C' \leftarrow \{h \cup \{H_i : v\} : h \in C, v \in \text{Domain}_i\}$

Prune:

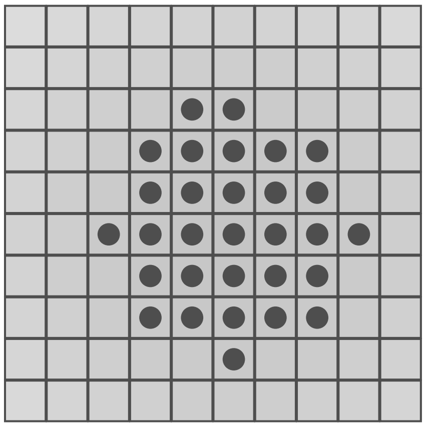
$C \leftarrow K$ particles of C' with highest weights

Normalize weights to get approximate $\hat{\mathbb{P}}(H_1, \dots, H_n \mid E = e)$

Sum probabilities to get any approximate $\hat{\mathbb{P}}(H_i \mid E = e)$

[demo: `beamSearch({K:3})`]

Beam search problems



Algorithm: beam search

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \ldots, n$:

Extend:

$C' \leftarrow \{h \cup \{H_i : v\} : h \in C, v \in \text{Domain}_i\}$

Prune:

$C \leftarrow K$ particles of C' with highest weights

- Extend: slow because requires considering every possible value for H_i
- Prune: greedily taking best K doesn't provide diversity

Particle filtering solution (3 steps): **propose, weight, resample**

Step 1: propose

Old particles: $\approx \mathbb{P}(H_1, H_2 \mid E_1 = 0, E_2 = 2)$

$\{H_1 : 0, H_2 : 1\}$
$\{H_1 : 1, H_2 : 2\}$

Key idea: proposal distribution

For each old particle (h_1, h_2), sample $H_3 \sim p(h_3 \mid h_2)$.

h_i	$p(h_i \mid h_{i-1})$
$h_{i-1} - 1$	1/4
h_{i-1}	1/2
$h_{i-1} + 1$	1/4

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2)$

$\{H_1 : 0, H_2 : 1, H_3 : 1\}$
$\{H_1 : 1, H_2 : 2, H_3 : 2\}$

Step 2: weight

Old particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 1)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$

Key idea: weighting based on evidence

For each old particle (h_1, h_2, h_3), weight it by $p(e_3 = 2 \mid h_3)$.

h_3	$p(e_3 = 2 \mid h_3)$
0	0
1	1/4
2	1/2

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 1, E_3 = 2)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ $(1/4)$
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ $(1/2)$

Step 3: resample

Old particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ (1/4) $\Rightarrow$ 1/3
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ (1/2) $\Rightarrow$ 2/3

Key idea: resampling

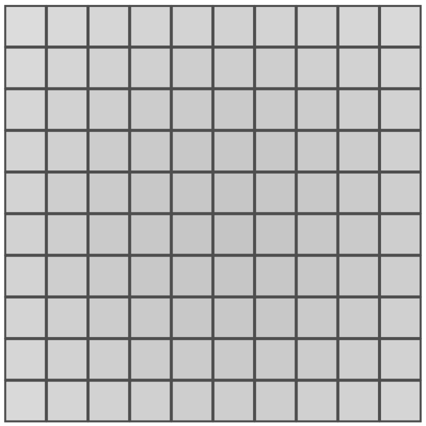
Normalize weights and draw K samples to redistribute particles to more promising areas.

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

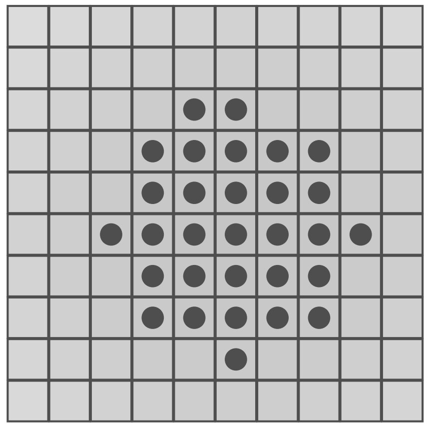
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$

Why sampling?

distribution

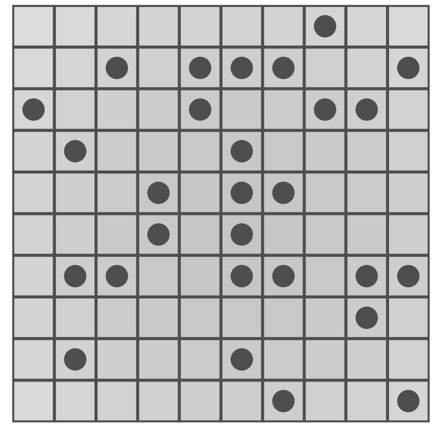


K with highest weight



not representative

K sampled from distribution



more representative

Sampling is especially important when there is high uncertainty!

Particle filtering

Algorithm: particle filtering

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \dots, n$:

Propose:

$$C' \leftarrow \{h \cup \{H_i : h_i\} : h \in C, h_i \sim p(h_i \mid h_{i-1})\}$$

Weight:

Compute weights $w(h) = p(e_i \mid h_i)$ for $h \in C'$

Resample:

$C \leftarrow K$ particles drawn independently from $\frac{w(h)}{\sum_{h' \in C} w(h')}$

[demo: `particleFiltering({K:100})`]

Particle filtering: implementation

For filtering questions, can optimize:

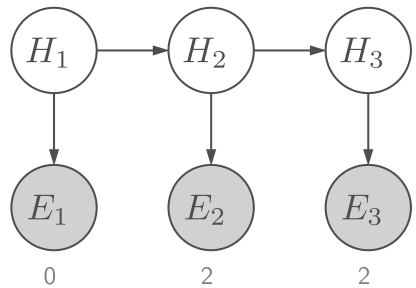
- Keep only value of last H_i for each particle
- Store count for each unique particle

$$
\begin{array}{lcccc}
\{H_1 : 0, H_2 : 1 : H_3 : 1\} & & 1 & & \\
\{H_1 : 0, H_2 : 1 : H_3 : 1\} & & 1 & & 1\ (2\text{x}) \\
\{H_1 : 1, H_2 : 2 : H_3 : 2\} & \longrightarrow & 2 & \longrightarrow & \\
\{H_1 : 1, H_2 : 1 : H_3 : 2\} & & 2 & & 2\ (3\text{x}) \\
\{H_1 : 1, H_2 : 2 : H_3 : 2\} & & 2 & &
\end{array}
$$

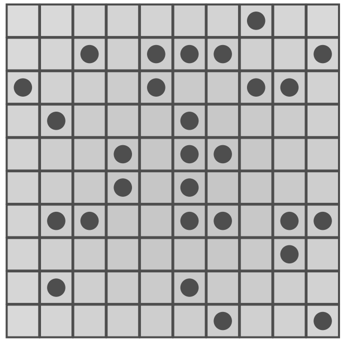
Particle filtering demo

[see web version]

Summary



$$\mathbb{P}(H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$$



- Use particles to represent an approximate distribution

Propose (transitions) **Weight** (emissions) **Resample**

- Can scale to large number of locations (unlike forward-backward)
- Maintains better particle diversity (compared to beam search)

Overall Summary: Bayesian Networks II

- Probabilistic programs as equivalent to Bayesian Networks
- Gibbs sampling is an algorithm for estimating marginal probabilities
- Forward Backward algorithm: Dynamic programming for inference (filtering and smoothing)
- Particle Filtering: Approximate inference for HMMs with large domains
- Next: learning the parameters of Bayesian networks