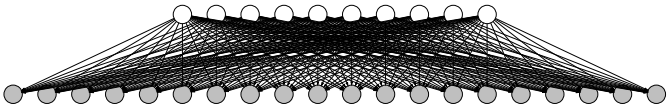
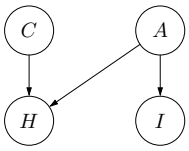
# Lecture 9.2: Bayesian networks III





# Review: Bayesian network




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

**Definition: Bayesian network**

Let $X = (X_1, \ldots, X_n)$ 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:

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



- A Bayesian network allows us to define a **joint** probability distribution over many variables (e.g., $\mathbb{P}(C, A, H, I)$) by specifying **local** conditional distributions (e.g., $p(i \mid a)$). Two lectures ago, we talked about modeling: how can we use Bayesian networks to represent real-world problems.

# Review: probabilistic inference

Bayesian network:

$$\mathbb{P}(X = x) = \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

Probabilistic inference:

$$\mathbb{P}(Q \mid E = e)$$

Algorithms:

- Variable elimination: general, exact
- Forward-backward: HMMs, exact



- Last lecture, we focused on probabilistic inference: how do we efficiently compute queries of interest? We showed that variable elimination can do many things in closed form by leveraging the conditional independence structure of Bayesian networks.
- For HMMs, we could use the forward-backward algorithm, which was just a special case of variable elimination, to compute the queries efficiently.
- For general Bayesian networks / factor graphs, variable elimination might be too slow. In this lecture, we will consider two **approximate** inference algorithms: Gibbs sampling and particle filtering.

# Roadmap

**Gibbs sampling**

Particle filtering

# Particle-based approximation

$$\mathbb{P}(X_1, X_2, X_3)$$

**Key idea: particles**

Use a small set of assignments (particles) to represent a large probability distribution.

| $x_1$ | $x_2$ | $x_3$ | $\mathbb{P}(X_1 = x_1, X_2 = x_2, X_3 = x_3)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.18 |
| 0 | 0 | 1 | 0.02 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0.08 |
| 1 | 1 | 1 | 0.72 |
| **0.8** | **0.8** | **0.74** | **(true marginals)** |

| | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| Sample 1 | 0 | 0 | 0 |
| Sample 2 | 1 | 1 | 1 |
| Sample 3 | 1 | 1 | 1 |
| **Estimated marginals** | **0.67** | **0.67** | **0.67** |



- The central idea to both of Gibbs sampling and particle filtering is the use of particles (just a fancy word for complete assignment, usually generated stochastically) to represent a probability distribution.
- Rather than storing the probability of every single assignment, we have a set of assignments, some of which can occur multiple times (which implicitly represents a higher probability). Maintaining this small set of assignments will allows us to answer queries more quickly, but at a cost: our answers will now be approximate instead of exact.
- From a set of particles, we can compute approximate marginals (or any query we want) by simply computing the fraction of assignments that satisfy the desired condition. Here, marginalization is easy because we're explicitly enumerating full assignments.
- Once we have a set of particles, we can compute all the queries we want with it. So now how do we actually generate the particles?

# Gibbs sampling

**Algorithm: Gibbs sampling**

Initialize $x$ to a random complete assignment

Loop through $i = 1, \ldots, n$ until convergence:

  Compute weight of $x \cup \{X_i : v\}$ for each $v$

  Choose $x \cup \{X_i : v\}$ with probability prop. to weight

**Gibbs sampling (probabilistic interpretation)**

Loop through $i = 1, \ldots, n$ until convergence:

  Set $X_i = v$ with prob. $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$

  (notation: $X_{-i} = X \backslash \{X_i\}$)

[demo]



- Recall that Gibbs sampling proceeds by going through each variable $X_i$, considering all the possible assignments of $X_i$ with some $v \in \text{Domain}_i$, computing the weight of the resulting assignment $x \cup \{X_i : v\}$, and choosing an assignment with probability proportional to the weight.
- We first introduced Gibbs sampling in the context of local search to get out of local optima. Now, we will use it for its original purpose, which is to draw samples from a probability distribution. (In particular, our goal is to draw from the joint distribution over $X_1, X_2, \ldots, X_n$.) To do this, we need to define a probability distribution given an arbitrary factor graph. Recall that a general factor graph defines a $\text{Weight}(x)$, which is the product of all its factors. We can simply normalize the weight to get a distribution: $\mathbb{P}(X = x) \propto \text{Weight}(x)$. Then, Gibbs sampling is exactly sampling according to the conditional distribution $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$.
- Note that the convergence criteria is on the *distribution* of the sample values, not the values themselves.
- Advanced: Gibbs sampling is an instance of a Markov Chain Monte Carlo (MCMC) algorithm which generates a sequence of particles $X^{(1)}, X^{(2)}, X^{(3)}, \ldots$. A Markov chain is irreducible if there is positive probability of getting from any assignment to any other assignment (now the probabilities are over the random choices of the sampler). When the Gibbs sampler is irreducible, then in the limit as $t \to \infty$, the distribution of $X^{(t)}$ converges to the true distribution $\mathbb{P}(X)$. MCMC is a very rich topic which we will not talk about very much here.
- In summary, Gibbs sampling is an algorithm that can be applied to any factor graph. It performs a guided random walk in the space of all possible assignments. Under some mild conditions, Gibbs sampling is guaranteed to converge to a sample from the true distribution $\mathbb{P}(X = x) \propto \text{Weight}(x)$ but this might take until the heat death of the universe for complex models. Nonetheless, Gibbs sampling is widely used due to its simplicity and can work reasonably well.

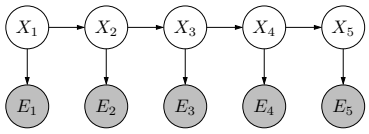
# Roadmap

Gibbs sampling

**Particle filtering**



# Hidden Markov models




$$\mathbb{P}(X = x, E = e) = \underbrace{p(x_1)}_{\text{start}} \prod_{i=2}^{n} \underbrace{p(x_i \mid x_{i-1})}_{\text{transition}} \prod_{i=1}^{n} \underbrace{p(e_i \mid x_i)}_{\text{emission}}$$

Query (**filtering**):

$$\mathbb{P}(X_1 \mid E_1 = e_1)$$

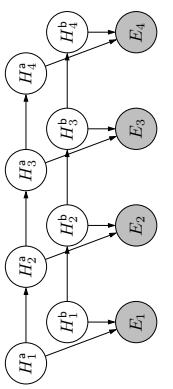
$$\mathbb{P}(X_2 \mid E_1 = e_1, E_2 = e_2)$$

$$\mathbb{P}(X_3 \mid E_1 = e_1, E_2 = e_2, E_3 = e_3)$$

## Factorial HMMs

**Probabilistic program: factorial HMM**

For each time step $t = 1, \ldots, T$:
  For each object $o \in \{\text{a}, \text{b}\}$:
    Generate location $H_t^o \sim p(H_t^o \mid H_{t-1}^o)$
  Generate sensor reading $E_t \sim p(E_t \mid H_t^{\text{a}}, H_t^{\text{b}})$








- Now we turn our attention to particle filtering. Gibbs sampling is the probabilistic analog of local search methods such as ICM, and particle filtering is the probabilistic analog of partial search such as beam search.
- Although particle filtering applies to general factor graphs, we will present them for hidden Markov models for concreteness. While forward-backward also does efficient inference in HMMs, particle filtering will naturally extend to more complex models like factorial HMMs, for which forward-backward can be expensive (since the total number of hidden states grows exponentially as you add more objects).
- As the name suggests, we will use particle filtering for answering filtering queries.

## Review: beam search

Idea: keep $\leq K$ **candidate list** $C$ of partial assignments

**Algorithm: beam search**

Initialize $C \leftarrow [\{\}]$
For each $i = 1, \ldots, n$:
  Extend:
    $C' \leftarrow \{x \cup \{X_i : v\} : x \in C, v \in \text{Domain}_i\}$
  Prune:
    $C \leftarrow K$ elements of $C'$ with highest weights

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



- Recall that beam search effectively does a pruned BFS of the search tree of partial assignments, where at each level, we keep track of the $K$ partial assignments with the highest weight.
- There are two phases. In the first phase, we **extend** all the existing candidates $C$ to all possible assignments to $X_i$; this results in $K = |\text{Domain}_i|$ candidates $C'$. These $C'$ are sorted by weight and **pruned** by taking the top $K$.

## Beam search

End result:

- Candidate list $C$ is set of particles
- Use $C$ to compute marginals

Problems:

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

Solution (3 steps): **propose, weight, resample**



- Beam search does generate a set of particles, but there are two problems.
- First, it can be slow if $\text{Domain}_i$ is large because we have to try every single value. Perhaps we can be smarter about which values to try.
- Second, we are greedily taking the top $K$ candidates, which can be too myopic. Can we somehow encourage more diversity?
- Particle filtering addresses both of these problems. There are three steps: propose, which extends the current parital assignment, and weight/resample, which redistributes resources on the particles based on evidence.

# Step 1: propose

Old particles: $\approx \mathbb{P}(X_1, X_2 \mid E_1 = 0, E_2 = 1)$

[0, 1]
[1, 0]

**Key idea: proposal distribution**

For each old particle $(x_1, x_2)$, sample $X_3 \sim p(x_3 \mid x_2)$.

New particles: $\approx \mathbb{P}(X_1, X_2, X_3 \mid E_1 = 0, E_2 = 1)$

[0, 1, 1]
[1, 0, 0]



- Suppose we have a set of particles that approximates the filtering distribution over $X_1, X_2$. The first step is to extend each current partial assignment (particle) from $x_{1:i-1} = (x_1, \ldots, x_{i-1})$ to $x_{1:i} = (x_1, \ldots, x_i)$.
- To do this, we simply go through each particle and extend it stocastically, using the transition probability $p(x_i \mid x_{i-1})$ to sample a new value of $X_i$.
- (For concreteness, think of what will happen if $p(x_i \mid x_{i-1}) = 0.8$ if $x_i = x_{i-1}$ and $0.2$ otherwise.)
- We can think of advancing each particle according to the dynamics of the HMM. These particles approximate the probability of $X_1, X_2, X_3$, but still conditioned on the same evidence.

# Step 2: weight

Old particles: $\approx \mathbb{P}(X_1, X_2, X_3 \mid E_1 = 0, E_2 = 1)$

[0, 1, 1]
[1, 0, 0]

**Key idea: weighting**

For each old particle $(x_1, x_2, x_3)$, weight it by $w(x_1, x_2, x_3) = p(e_3 \mid x_3)$.

New particles:

$$\approx \mathbb{P}(X_1, X_2, X_3 \mid E_1 = 0, E_2 = 1, E_3 = 1)$$

[0, 1, 1] (0.8)
[1, 0, 0] (0.4)



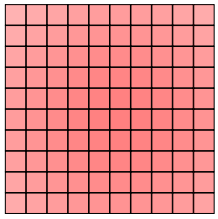
- Having generated a set of $K$ candidates, we need to now take into account the new evidence $E_i = e_i$. This is a deterministic step that simply weights each particle by the probability of generating $E_i = e_i$, which is the emission probability $p(e_i \mid x_i)$.
- Intuitively, the proposal was just a guess about where the object will be $X_3$. To get a more realistic picture, we condition on $E_3 = 1$. Supposing that $p(e_i = 1 \mid x_i = 1) = 0.8$ and $p(e_i = 1 \mid x_i = 0) = 0.4$, we then get the weights 0.8 and 0.4. Note that these weights do not sum to 1.

# Step 3: resample

Question: given weighted particles, which to choose?

Tricky situation:

- Target distribution close to uniform
- Fewer particles than locations

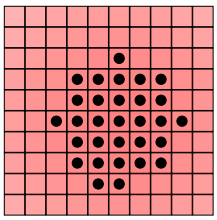




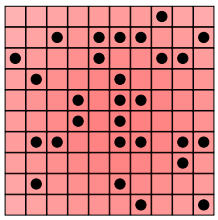
- Having proposed extensions to the particles and computed a weight for each particle, we now come to the question of which particles to keep.
- Intuitively, if a particle has very small weight, then we might want to prune it away. On the other hand, if a particle has high weight, maybe we should dedicate more resources to it.
- As a motivating example, consider an almost uniform distribution over a set of locations, and trying to represent this distribution with fewer particles than locations. This is a tough situation to be in.

# Step 3: resample

$K$ with highest weight



$K$ sampled from distribution



Intuition: top $K$ assignments not representative.

Maybe random samples will be more representative...



- Beam search, which would choose the $K$ locations with the highest weight, would clump all the particles near the mode. This is risky, because we have no support out farther from the center, where there is actually substantial probability.
- However, if we sample from the distribution which is proportional to the weights, then we can hedge our bets and get a more representative set of particles which cover the space more evenly.

# Step 3: resample

**Key idea: resampling**

Given a distribution $\mathbb{P}(A = a)$ with $n$ possible values, draw a sample $K$ times.

Intuition: redistribute particles to more promising areas

**Example: resampling**

| $a$ | $\mathbb{P}(A = a)$ |
|---|---|
| a1 | 0.70 |
| a2 | 0.20 |
| a3 | 0.05 |
| a4 | 0.05 |

| | |
|---|---|
| sample 1 | a1 |
| sample 2 | a2 |
| sample 3 | a1 |
| sample 4 | a1 |



- After proposing and weighting, we end up with a set of samples $x_{1:i}$, each with some weight $w(x_{1:i})$. Intuitively, if $w(x_{1:i})$ is really small, then it might not be worth keeping that particle around.
- Resampling allows us to put (possibly multiple) particles on high weight particles. In the example above, we don't sample a3 and a4 because they have low probability of being sampled.

# Step 3: resample

Old particles:

$$\approx \mathbb{P}(X_1, X_2, X_3 \mid E_1 = 0, E_2 = 1, E_3 = 1)$$

[0, 1, 1] (0.8) $\Rightarrow 2/3$
[1, 0, 0] (0.4) $\Rightarrow 1/3$

New particles:

$$\approx \mathbb{P}(X_1, X_2, X_3 \mid E_1 = 0, E_2 = 1, E_3 = 1)$$

[0, 1, 1]
[0, 1, 1]



- In our example, we normalize the particle weights to form a distribution over particles. Then we draw $K = 2$ independent samples from that distribution. Higher weight particles will get more samples.

# Particle filtering

**Algorithm: particle filtering**

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

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

Propose (extend):

$C' \leftarrow \{x \cup \{X_i : x_i\} : x \in C, x_i \sim p(x_i \mid x_{i-1})\}$

Reweight:

Compute weights $w(x) = p(e_i \mid x_i)$ for $x \in C'$

Resample (prune):

$C \leftarrow K$ elements drawn independently from $\propto w(x)$

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



- The final algorithm here is very similar to beam search. We go through all the variables $X_1, \ldots, X_n$.
- For each candidate $x_{i-1} \in C$, we propose $x_i$ according to the transition distribution $p(x_i \mid x_{i-1})$.
- We then weight this particle using $w(x) = p(e_i \mid x_i)$.
- Finally, we select $K$ particles from $\propto w(x)$ by sampling $K$ times independently.

# Particle filtering: implementation

- If only care about last $X_i$, collapse all particles with same $X_i$ (think elimination)

$$001 \Rightarrow 1$$
$$101 \Rightarrow 1$$
$$010 \Rightarrow 0$$
$$010 \Rightarrow 0$$
$$110 \Rightarrow 0$$

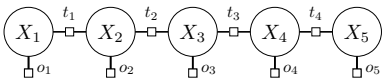
- If many particles are the same, can just store counts

$$\begin{matrix}1\\1\\0\\0\\0\end{matrix} \Rightarrow \begin{matrix}1:2\\0:3\end{matrix}$$



- In particle filtering as it is currently defined, each particle is an entire trajectory in the context of object tracking (assignment to all the variables).
- Often in tracking applications, we only care about the last location $X_i$, and the HMM is such that the future $(X_{i+1}, \ldots, X_n)$ is conditionally independent of $X_1, \ldots, X_{i-1}$ given $X_i$. Therefore, we often just store the value of $X_i$ rather than its entire ancestry.
- When we only keep track of the $X_i$, we might have many particles that have the same value, so it can be useful to store just the counts of each value rather than having duplicates.

# Application: tracking




**Example: tracking**

- $X_i$: position of object at $i$
- Transitions: $t_i(x_i, x_{i+1}) = [x_i \text{ near } x_{i+1}]$
- Observations: $o_i(x_i) =$ sensor reading...



# Particle filtering demo

[see web version]

- Consider a tracking application where an object is moving around in a grid and we are trying to figure out its location $X_i \in \{1, \ldots, \text{grid-width}\} \times \{1, \ldots, \text{grid-height}\}$.
- The transition factors say that from one time step to the next, the object is equally likely to have moved north, south, east, west, or stayed put.
- Each observation is a location on the grid (a yellow dot). The observation factor is a user-defined function which depends on the vertical and horizontal distance.
- Play around with the demo to get a sense of how particle filtering works, especially the different observation factors.

# Probabilistic inference

Model (Bayesian network or factor graph):

$$\mathbb{P}(X = x) = \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

Probabilistic inference:

$$\mathbb{P}(Q \mid E = e)$$

Algorithms:

- Forward-backward: chain-structured (HMMs), exact
- Gibbs sampling, particle filtering: general, approximate

Next time: learning