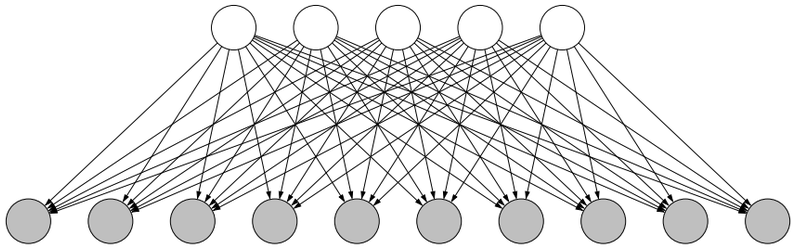
# Bayes nets: HMM inference



- In this lecture, I will introduce the forward-backward algorithm for performing efficient and exact inference in Hidden Markov models, an important special case of Bayesian networks.

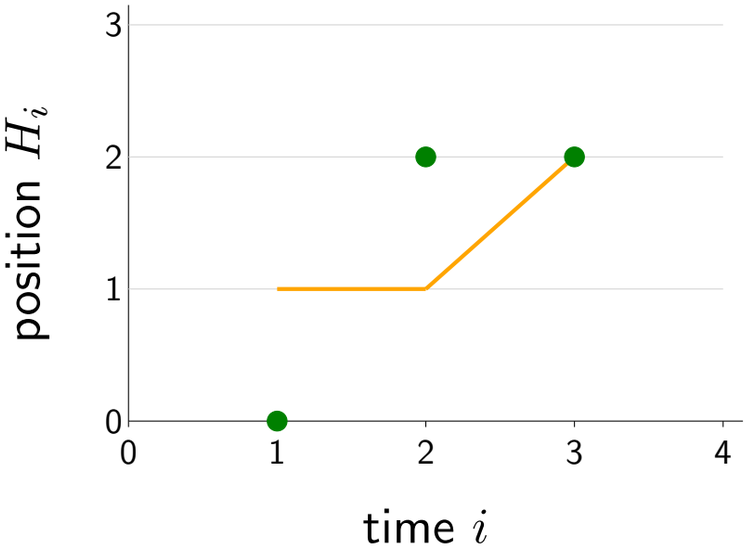
# Roadmap

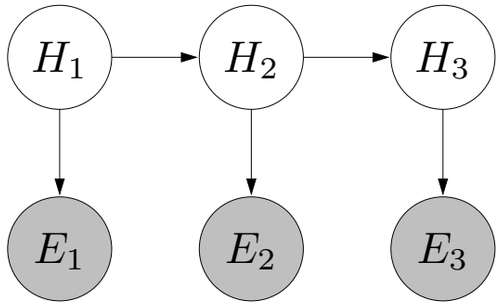
**Hidden Markov Models (c15)**

Forward Backward (c15.2)

Particle Filter and Gibbs (c15.5.3)

# Hidden Markov models for object tracking

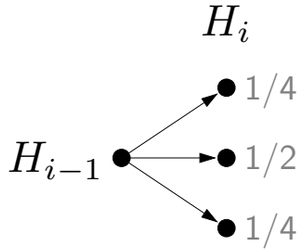



start

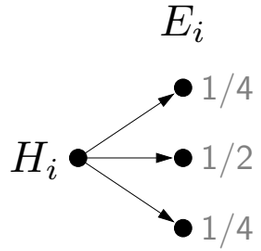

| $h_1$ | $p(h_1)$ |
|---|---|
| 0 | 1/3 |
| 1 | 1/3 |
| 2 | 1/3 |

transition




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

emission

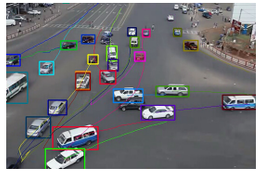



| $e_i$ | $p(e_i \mid h_i)$ |
|---|---|
| $h_i-1$ | 1/4 |
| $h_i$ | 1/2 |
| $h_i+1$ | 1/4 |

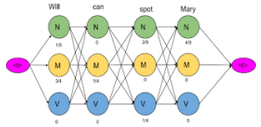
$$\mathbb{P}(H=h, E=e) = \underbrace{p(h_1)}_{\text{start}} \prod_{i=2}^{n} \underbrace{p(h_i \mid h_{i-1})}_{\text{transition}} \prod_{i=1}^{n} \underbrace{p(e_i \mid h_i)}_{\text{emission}}$$

- Let us revisit our object tracking example, now through the lens of HMMs. Recall that each time $i$, an object is at a location $H_i$, and what we observe is a noisy observation $E_i$. The goal is to infer where the object is / was.
- We define a probabilistic story as follows: An object starts at $H_1$ uniformly drawn over all possible locations.
- Then at each subsequent time step, the object **transitions** from the previous time step, keeping the same location with $1/2$ probability, and moves to an adjacent location each with $1/4$ probability. For example, if $p(h_3 = 3 \mid h_2 = 3) = 1/2$ and $p(h_3 = 2 \mid h_2 = 3) = 1/4$.
- At each time step, we also **emit** a sensor reading $E_i$ given the actual location $H_i$, following the same process as transitions ($1/2$ probability of the same location, $1/4$ probability of an adjacent location).
- Recall that finally, we define a joint distribution over all the actual locations $H_1, \ldots, H_n$ and sensor readings $E_1, \ldots, E_n$ by taking the product of all the local conditional probabilities.

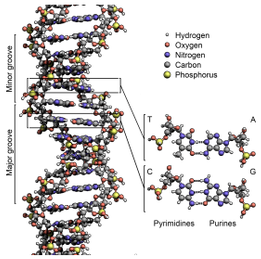
# Applications



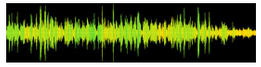
Object tracking: tracking cars (driving), rockets (kalman filters).




Part of speech tagging: labeling a sentence with its part of speech.

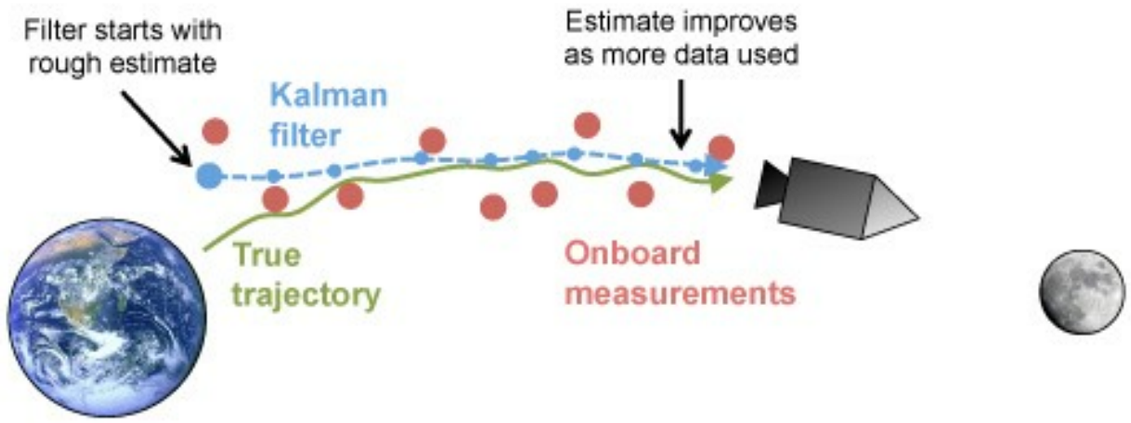



Genomics: identifying gene-coding regions in the genome.



Speech recognition: recovering text (latent) from speech.

# HMMs (kalman filter) in space!




As mentioned earlier, the Kalman filter has been used in a variety of fields. Recently, a special issue of the IEEE Transations on Automatic Control (ref. 38) was devoted to papers on applications that were as wide ranging as possible in their subject matter. The papers cover such diverse subjects as spacecraft orbit determination, prediction of cattle populations in France, radar tracking, navigation, ship motion, natural gamma ray spectroscopy in oil- and gas-well exploration, measurement of instantaneous flow rates, and estimation and prediction of unmeasurable variables in industrial processes, on-line failure detection in nuclear plant instrumentation, and power station control systems. In many cases, the solutions in these papers were implemented and were operationally successful. Indeed, the broad application of the filter to seemingly unlikely problems suggests that we have only scratched the surface when it comes to possible applications, and that we will likely be amazed at the applications to which this filter will be put in the years to come.

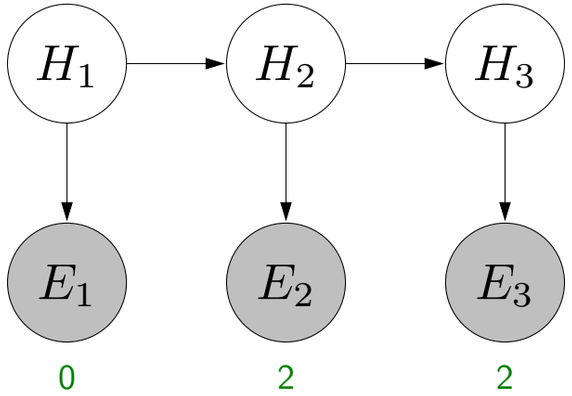
# Roadmap

Hidden Markov Models (c15)

**Forward Backward (c15.2)**

Particle Filter and Gibbs (c15.5.3)

# Inference questions




Question (**filtering**):

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

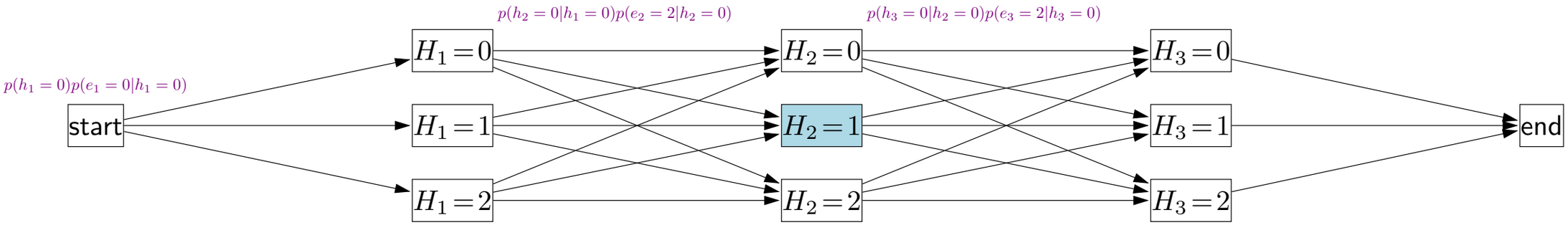
Question (**smoothing**):

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

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

- In principle, you could ask any type of questions on an HMM, but there are two common ones: filtering and smoothing.
- **Filtering** asks for the distribution of some hidden variable $H_i$ conditioned on only the evidence up until that point. This is useful when you're doing real-time object tracking, and you can't see the future.
- **Smoothing** asks for the distribution of some hidden variable $H_i$ conditioned on all the evidence, including the future. This is useful when you have collected all the data and want to retrospectively go and figure out what $H_i$ was.
- Note that filtering is a special case of smoothing: if we're asking for $H_i$ given $E_1, \ldots, E_i$, then we can marginalize everything in the future (since they are just unobserved leaf nodes), reducing the problem to a smaller HMM, where we are smoothing.

# Lattice representation

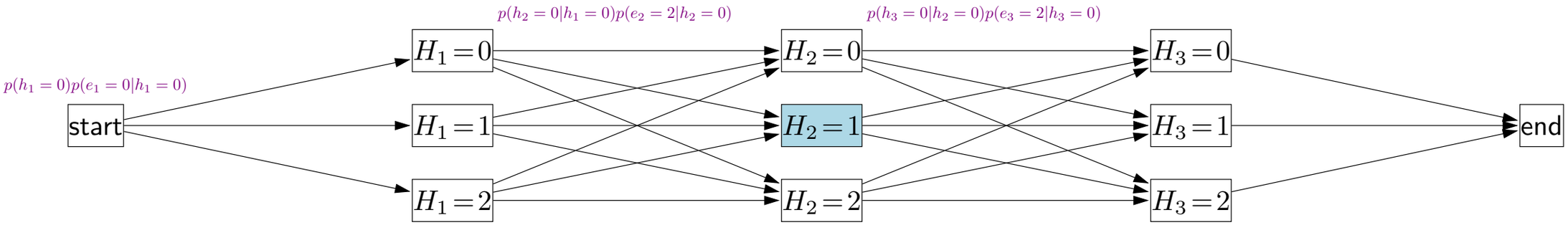



- Edge $\boxed{\text{start}} \Rightarrow \boxed{H_1 = h_1}$ has weight $p(h_1)p(e_1 \mid h_1)$
- Edge $\boxed{H_{i-1} = h_{i-1}} \Rightarrow \boxed{H_i = h_i}$ has weight $p(h_i \mid h_{i-1})p(e_i \mid h_i)$
- Each path from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ is an assignment with weight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

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

- The forward-backward algorithm is based on a form of dynamic programming.
- To develop this, we consider a **lattice representation** of HMMs. Consider a directed graph (not to be confused with the HMM) with a start node, an end node, and a node for each assignment of a value to a variable $H_i = v$. The nodes are arranged in a lattice, where each column corresponds to one variable $H_i$ and each row corresponds to a particular value $v$. Each path from the start to the end corresponds exactly to a complete assignment to the nodes.
- Each edge has a weight (a single number) determined by the local conditional probabilities (more generally, the factors in a factor graph). For each edge into $\boxed{H_i = h_i}$, we multiply by the transition probability into $h_i$ and emission probability $p(e_i \mid h_i)$.
- This defines a weight for each path (assignment) in the graph equal to the joint probability $P(H = h, E = e)$.
- Note that the lattice contains $O(n|\text{Domain}|)$ nodes and $O(n|\text{Domain}|^2)$ edges, where $n$ is the number of variables and $|\text{Domain}|$ is the number of values in the domain of each variable (3 in our example).
- Now comes the key point. Recall we want to compute a smoothing question $\mathbb{P}(H_i = h_i \mid E = e)$. This quantity is simply the weighted fraction of paths that pass through $\boxed{H_i = h_i}$. This is just a way of visualizing the definition of the smoothing question.
- There are an exponential number of paths, so it's intractable to enumerate all of them. But we can use dynamic programming...

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

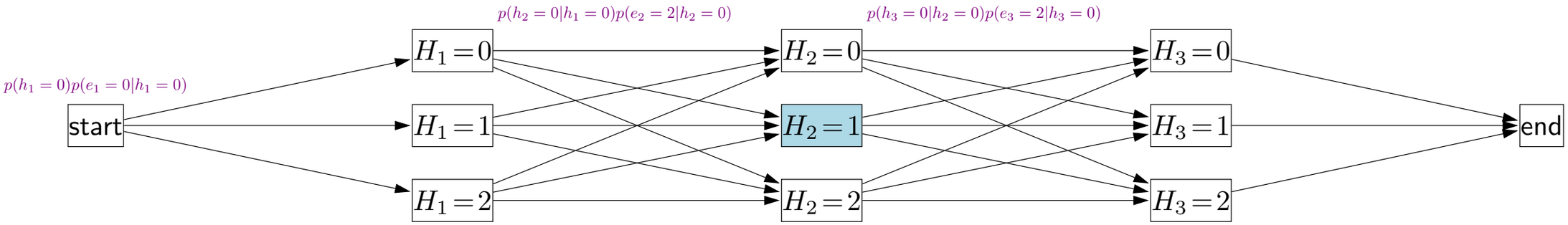
sum of weights of paths from $\boxed{H_i = h_i}$ to $\boxed{\text{end}}$

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 $\boxed{H_i = h_i}$

- First, define the forward message $F_i(v)$ to be the sum of the weights over all paths from the start node to $\boxed{H_i = v}$. This can be defined recursively: any path that goes $\boxed{H_i = h_i}$ will have to go through some $\boxed{H_{i-1} = h_{i-1}}$, so we can sum over all possible values of $h_{i-1}$.
- Analogously, let the backward message $B_i(v)$ be the sum of the weights over all paths from $\boxed{H_i = v}$ to the end node.
- Finally, define $S_i(v)$ to be the sum of the weights over all paths from the start node to the end node that pass through the intermediate node $\boxed{X_i = v}$. This quantity is just the product of the weights of paths going into $\boxed{H_i = h_i}$ $(F_i(h_i))$ and those leaving it $(B_i(h_i))$.
- This is analogous to factoring: $(a + b)(c + d) = ab + ad + bc + bd$.
- Note: $F_1(h_1) = p(h_1)p(e_1 = 0 \mid h_1)$ and $B_n(h_n) = 1$ are base cases, which don't require the recurrence.

# 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 $F_1, F_2, \ldots, F_n$

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

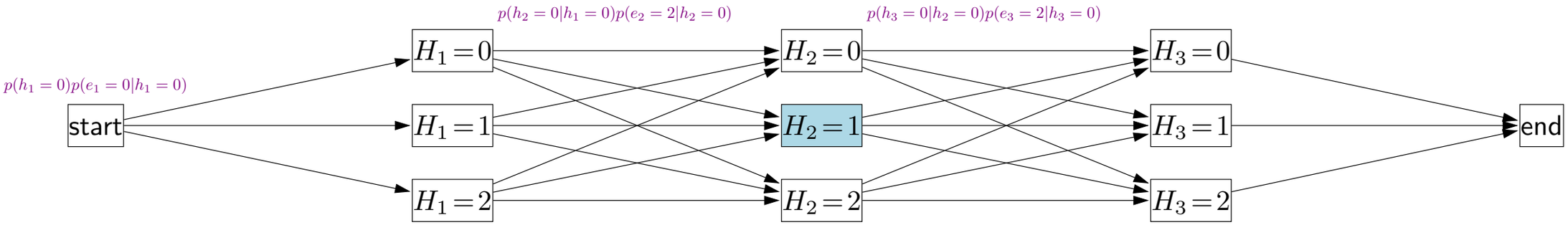
Compute $S_i$ for each $i$ and normalize

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

[demo]

- Now the smoothing question $\mathbb{P}(H_i = h_i \mid E = e)$ is just equal to the normalized version of $S_i$.
- The algorithm is thus as follows: for each node $\boxed{H_i = h_i}$, we compute three numbers: $F_i(h_i), B_i(h_i), S_i(h_i)$. First, we sweep forward to compute all the $F_i$'s recursively. At the same time, we sweep backward to compute all the $B_i$'s recursively. Then we compute $S_i$ by pointwise multiplication.
- The running time of the algorithm is $O(n|\text{Domain}|^2)$, which is the number of edges in the lattice.
- In the demo, we are running the variable elimination algorithm, which is a generalization of the forward-backward algorithm for arbitrary Markov networks. As you step through the algorithm, you can see that the algorithm first computes a forward message $F_2$ and then a backward message $B_2$, and then it multiplies everything together and normalizes to produce $\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2, E_3 = 2)$. The names and details don't match up exactly, so you don't need to look too closely.
- Implementation note: we technically can normalize $S_i$ to get $\mathbb{P}(H_i \mid E = e)$ at the very end but it's useful to normalize $F_i$ and $B_i$ at each step to avoid underflow. In addition, normalization of the forward messages yields $\mathbb{P}(H_i = v \mid E_1 = e_1, \ldots, E_i = e_i)$ which are exactly the filtering queries!

# Summary




- Lattice representation: paths are assignments
- Dynamic programming: compute sums efficiently
- Forward-backward algorithm: compute all smoothing questions, share intermediate computations

- In summary, we have presented the forward-backward algorithm for probabilistic inference in HMMs, in particular smoothing queries.
- The algorithm is based on the lattice representation in which each path is an assignment, and the weight of path is the joint probability.
- Smoothing is just then asking for the weighted fraction of paths that pass through a given node.
- Dynamic programming can be used to compute this quantity efficiently.
- This is formalized using the forward-backward algorithm, which consists of two sets of recurrences.
- Note that the forward-backward algorithm gives you the answer to all the smoothing questions ($\mathbb{P}(H_i = h_i \mid E = e)$ for all $i$), because the intermediate computations are all shared.

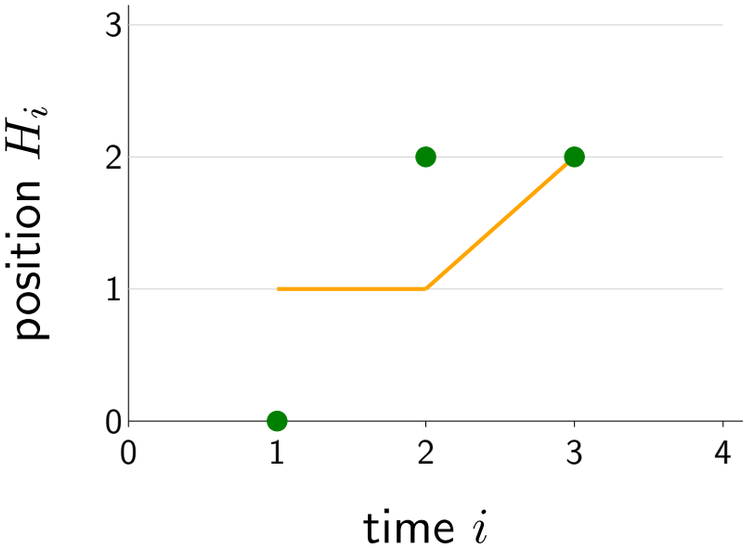
# Roadmap

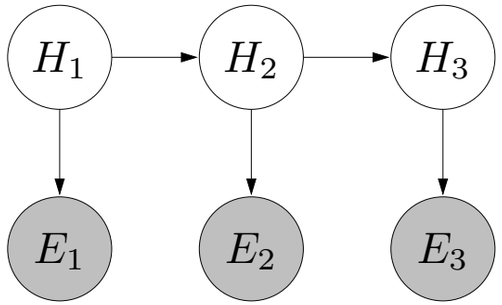
Hidden Markov Models (c15)

Forward Backward (c15.2)

**Particle Filter and Gibbs (c15.5.3)**

# Review: Hidden Markov models for object tracking

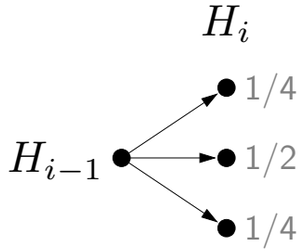



start

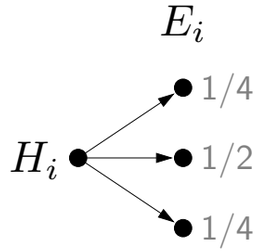

| $h_1$ | $p(h_1)$ |
|---|---|
| 0 | 1/3 |
| 1 | 1/3 |
| 2 | 1/3 |

transition




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

emission

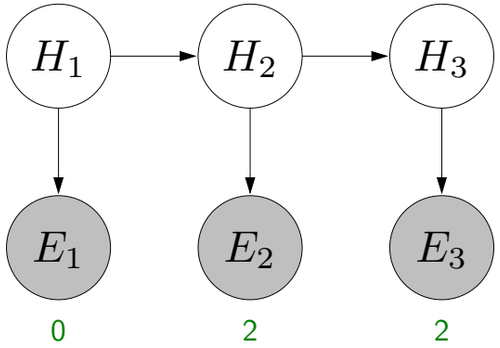



| $e_i$ | $p(e_i \mid h_i)$ |
|---|---|
| $h_i-1$ | 1/4 |
| $h_i$ | 1/2 |
| $h_i+1$ | 1/4 |

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

- Recall that HMM for object tracking.
- Each each point in time, an object has an position $H_i$, which gives rise to a sensor reading $E_i$. We start with $H_1$ uniform over positions, transition from $H_{i-1}$ to $H_i$ with $1/2$ probability on the same location and $1/4$ probability on an adjacent location. We emit the sensor reading analogously. Multiply everything together to form the joint distribution over locations $H_1, \ldots, H_n$ and sensor readings $E_1, \ldots, E_n$.

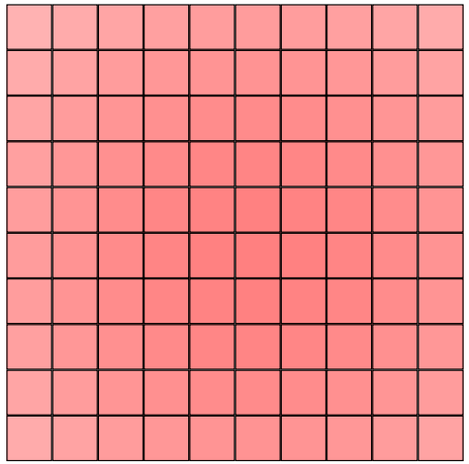
# Review: inference in Hidden Markov models




**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 $H_i$

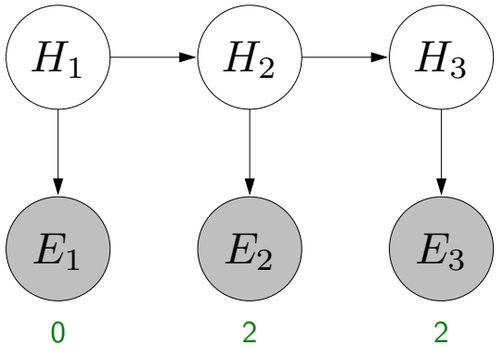


Forward-backward is too slow ($O(n|\text{Domain}|^2)$)...

- Recall that the two common types of inference questions we ask on HMMs are filtering and smoothing.
- Particle filtering, as the name might suggest, performs filtering, so let us focus on that. Filtering asks for the probability distribution over object location $H_i$ at a current time step $i$ given the past observations $E_1 = e_1, \ldots, E_i = e_i$.
- Last time, we saw that the forward-backward algorithm could already solve this. But it runs in $O(n|\text{Domain}|^2)$, where $|\text{Domain}|$ is the number of possible values (e.g., locations) that $H_i$ can take on. On this example, $H_i \in \{0, 1, 2\}$ but for real applications, there could easily be hundreds of thousands of values, not to mention what happens if $H_i$ is continuous. This could be a very large number, which makes the forward-backward algorithm very slow (even if it's not exponentially so).
- The motivation of particle filtering is to perform **approximate probabilistic inference**, and leverages the fact that most of the locations are very improbable given evidence.
- Particle filtering actually applies to general factor graphs, but we will present them for hidden Markov models for concreteness.

# Beam search for HMMs

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




**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

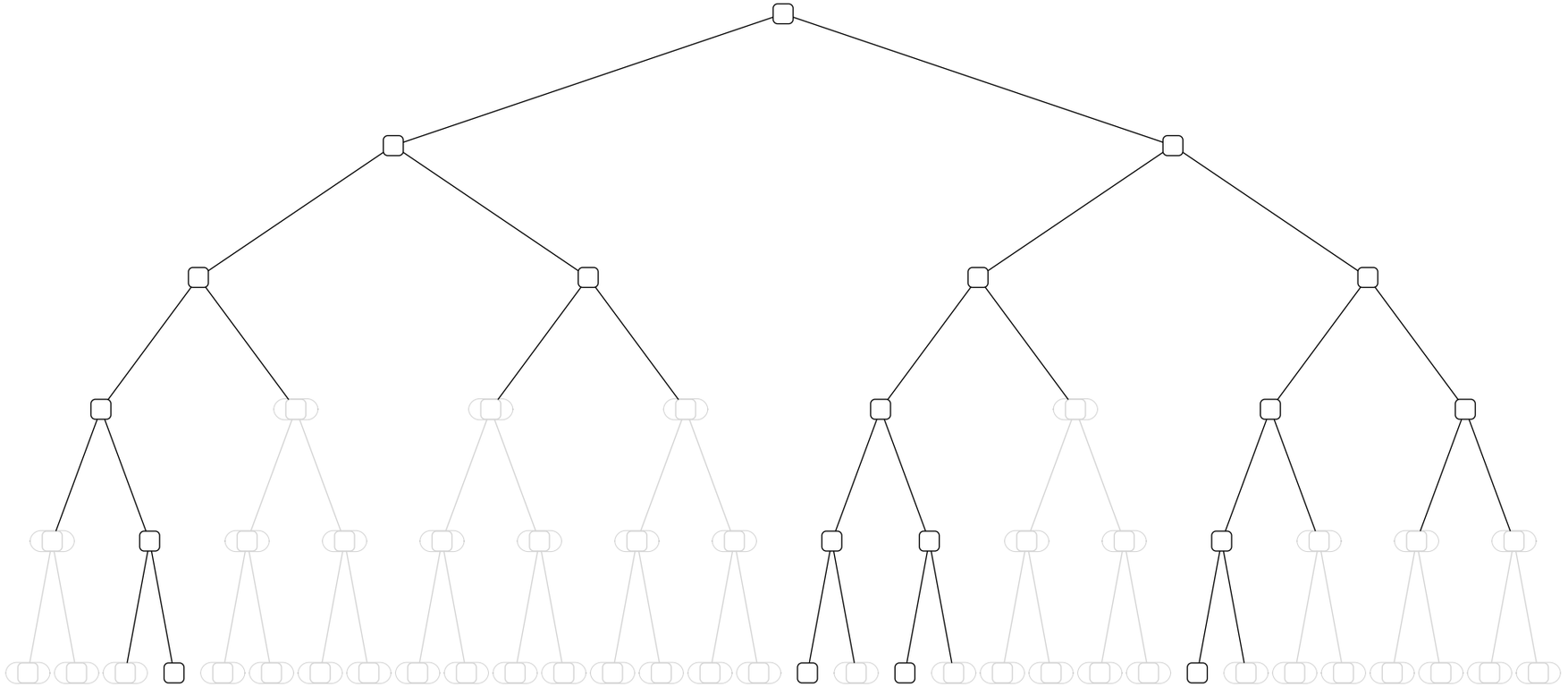
Normalize weights to get approximate $\hat{\mathbb{P}}(H_1, \ldots, H_n \mid E = e)$

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

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

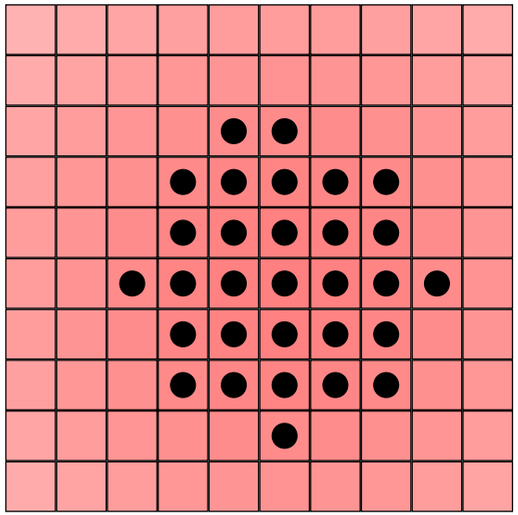
- Our starting point for motivating particle filtering is beam search, an algorithm for finding an approximate maximum weight assignment in arbitrary constraint satisfaction problems (CSPs).
- Since HMMs are Bayesian networks, which are Markov networks, which have an underlying factor graph, we can simply apply beam search to HMMs (for now putting aside the goal of finding the maximum weight assignment).
- Recall that beam search maintains a list of candidate partial assignments to the first $i$ variables. There are two phases. In the first phase, we **extend** all the existing candidates $C$ to all possible assignments to $H_i$; this results in $K = |\text{Domain}|$ candidates $C'$. We then take the subset of $K$ candidates with the highest weight, where the weight of a partial assignment is simply the product of all the factors (transitions, emissions) that can be computed on the partial assignment.
- In the demo, we start with partial assignments to $H_1$, whose weights are given by $p(h_1)p(e_1 = 0 \mid h_1)$. In the next step, we can multiply in factors $p(h_2 \mid h_1)p(e_2 = 2 \mid h_2)$, and so on.
- At the very end, we obtain $K = 3$ complete assignments, each with a weight (equal to the joint probability of the assignment and observations). We can normalize these weights to form an approximate distribution over all assignments (conditioned on the observations). From here, we can manually compute any marginal probabilities (e.g., $\mathbb{P}(H_3 = 2 \mid E = e)$) by summing the probabilities of assignments satisfying the given condition (e.g., $H_3 = 2$).

# Review: Beam search



Beam size $K = 4$

# 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**

- There are two problems with beam search.
- First, beam search can be slow if Domain is large, since we might have to try every single candidate value $h_i$ to assign $H_i$. In some cases, we can efficiently generate only the values $h_i$ that have nonzero transition probability ($p(h_i \mid h_{i-1} > 0)$, for example, if we know that $h_i$ must be within a certain distance of $h_{i-1}$ (can't teleport). But if we wanted to track the object to high resolution, there might still be too many values to consider.
- Second, beam search greedily takes the $K$ highest weight candidates at each time step. This could be dangerous, since we might end up with many assignments that are only slightly different, and not truly representative of the actual distribution. You can think of this as a form of overfitting.
- Particle filtering addresses both of these problems. It has three steps: propose, which extends the current partial assignment, and reweight + resample, which redistributes resources on the particles based on evidence.

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

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

- At each stage of the particle filtering, we can think of our set of particles $C$ as approximating a certain distribution.
- Suppose we have a set of particles that approximates the filtering distribution over $H_1, H_2$. The first step is to extend each current partial assignment (particle) from $(h_1, \ldots, h_{i-1})$ to $(h_1, \ldots, h_i)$.
- To do this, we simply go through each particle and sample a new value $h_i$ using the transition probability $p(h_i \mid h_{i-1})$.
- We can think of advancing each particle according to the dynamics of the HMM. These extended particles approximate the probability of $H_1, H_2, H_3$, but still conditioned on the same evidence.
- In some cases (e.g., the transitions are Gaussian), sampling $h_3$ is very easy compared to enumerating all possible of $h_3$. (Indeed, the advantages of particle filtering are clearer in continuous state spaces.).

# 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)

- 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 $p(e_i \mid h_i)$.
- Intuitively, the proposal was just a guess about where the object will be $H_3$, but we need to fact check this guess.
- In this example, we observed $E_3 = 2$, so we need to weight the two particles by $p(e_3 = 2 \mid h_3 = 1) = 1/4$ and $p(e_3 = 2 \mid h_3 = 2) = 1/2$, respectively.

# 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.

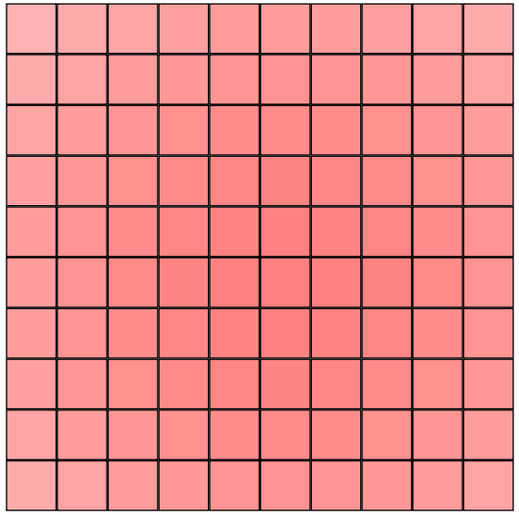
New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

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

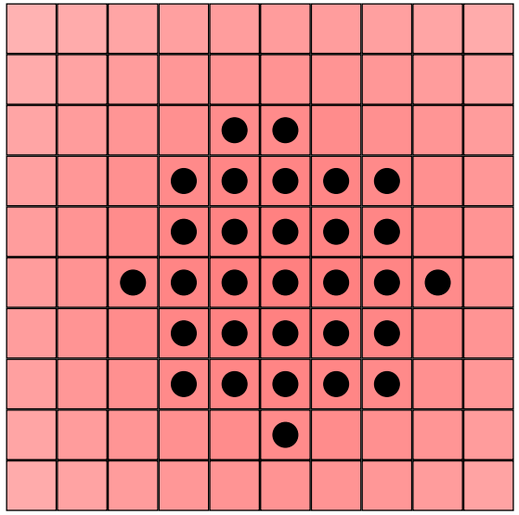
- At this point, we have a set of weighted particles representing the desired filtering distribution.
- However, if some of the weights are small, this could be wasteful. In the extreme case, any particle with zero weight should just be thrown out.
- The $K$ particles can be viewed as our limited resources for representing the distribution, the resampling step attempts to redistribute these precious resources to places in the distribution that are more promising.
- To this end, we will normalize the weights to form a distribution over the particles (similar to what we did at the end of beam search). Then we sample $K$ times from this distribution.
- In this example, we happened to get two occurrences of the second particle, but we might have easily gotten one of each or even two of the first.

# Why sampling?

distribution

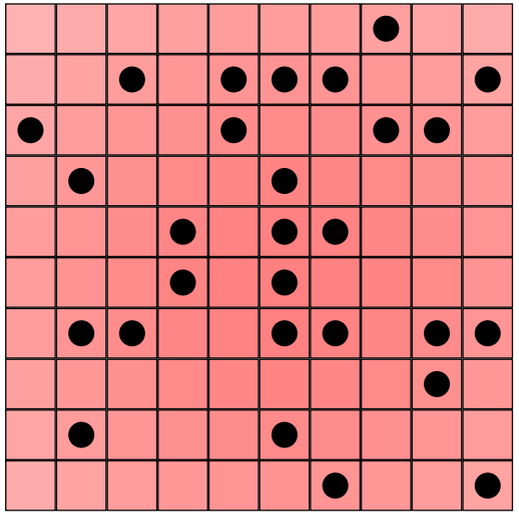


$K$ with highest weight



not representative

$K$ sampled from distribution



more representative

Sampling is especially important when there is high uncertainty!

- You might wonder why we are resampling, leaving the result of the algorithm up to chance.
- To see why resampling can be more favorable than beam search, consider the setting where we start with a set of particles on the left where the weights are given by the shade of red (darker is more weight). Notice that the weights are all quite similar (i.e., the distribution is close to the uniform distribution).
- Beam search chooses the $K$ locations with the highest weight, which 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.
- In cases where the original weights much more skewed towards a few particles, then taking the highest weight particles is fine and perhaps even slightly better than resampling.

# Particle filtering

**Algorithm: particle filtering**

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

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

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 $\frac{w(h)}{\sum_{h' \in C} w(h')}$

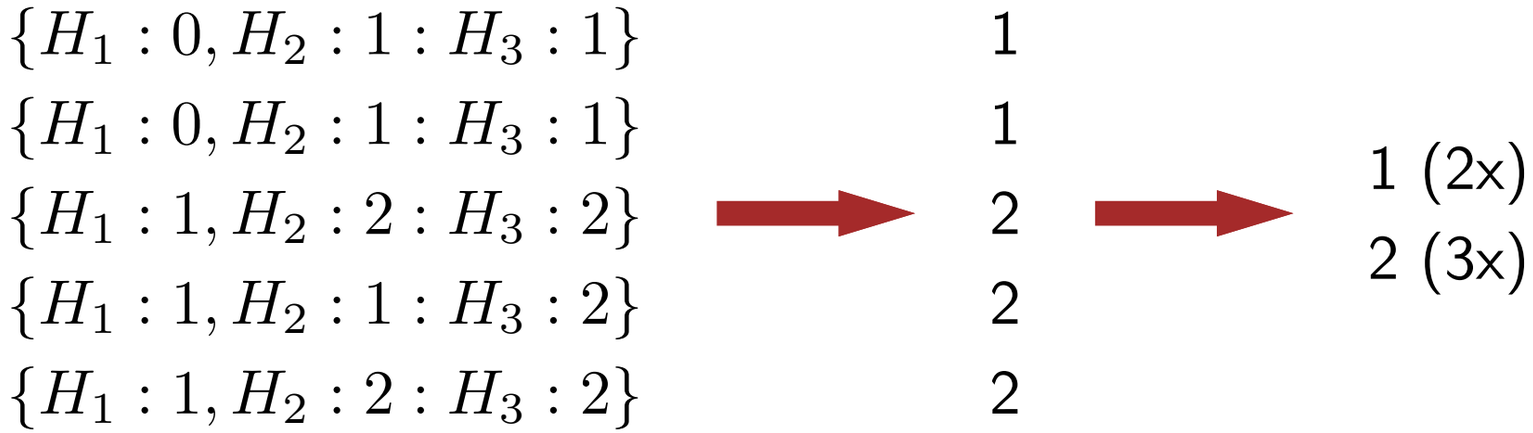
[demo: `particleFiltering({K:100})`]

- We now present the final particle filtering algorithm, which is structurally similar to beam search. We go through all the variables $H_1, \ldots, H_n$.
- For each candidate $h \in C$, we propose $h_i$ according to the transition distribution $p(h_i \mid h_{i-1})$.
- We then weight this particle using the emission probability $w(h) = p(e_i \mid h_i)$.
- Finally, we normalize the weights $\{w(h) : h \in C\}$ and sample $K$ particles independently from this distribution.
- In the demo, we can go through the extend (propose) and prune (weight + resample) steps, ending with a final set of full assignments, which can be used to approximate the filtering distribution $\mathbb{P}(H_3 \mid E = e)$.

# Particle filtering: implementation

For filtering questions, can optimize:

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



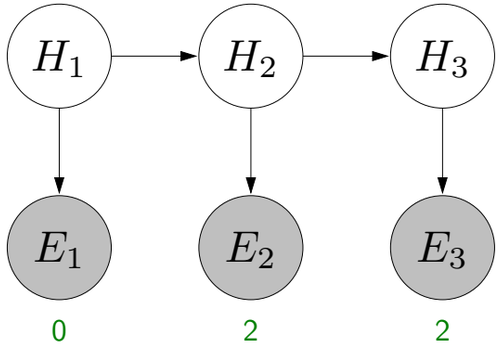
- So far, we have presented a version of particle filtering where each particle at the end is a full assignment to all the variables. This allows us to approximately answer a variety of different questions based on the induced distribution.
- However, if we're only interested in filtering questions, then we can perform two optimizations.
- First, in tracking applications, we only care about the last location $H_i$, and future steps only depend on the value of $H_i$. Therefore, we often just store the value of $H_i$ rather than the entire trajectory.
- Second, since we have discrete variables, many particles might have the same value of $H_i$, so we can just store the counts of each value rather than storing duplicate values.

# Particle filtering demo

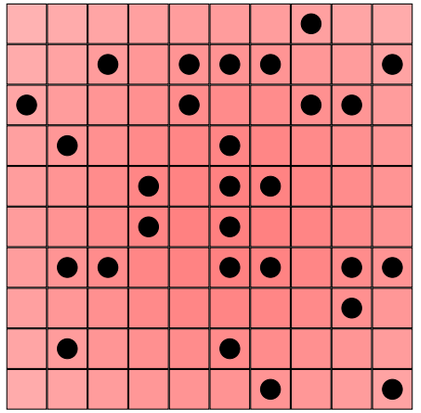
[see web version]

- Now let us visualize particle filtering in a more realistic, interactive object tracking setting.
- Consider an object is moving around in a grid and we are trying to figure out its location $H_i \in \{1, \ldots, \text{grid-width}\} \times \{1, \ldots, \text{grid-height}\}$.
- The transition distribution places a uniform distribution over moving north, moving south, moving east, moving west, or staying put.
- The emission distribution places a uniform distribution over locations $E_i$ that are within 3 steps (both vertically and horizontally) of the actual position $H_i$. In the textbox, you can change the emission distribution dynamically (`observeFactor`).
- When you hit ctrl-enter, you can see the noisy sensor readings (visualized as a yellow dot bouncing around).
- If you increase the number of particles, you can see a red cloud representing where the particles are, where the intensity of a square is proportional to the number of particles in that square.
- You can now set `showTruePosition = true` to see the actual $H_i$ that generated $E_i$. You can see that the cloud is able to track the true location reasonably well, although there are occasional errors.

# 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)

- In summary, we have presented particle filtering, an inference algorithm for HMMs that approximately computes filtering questions of the form: where is the object currently given all the past noisy sensor readings?
- Particle filtering represents distributions over hidden variables with a set of particles. To advance the particles to the next time step, it proposes new positions based on transition probabilities. It then weights these guesses based on evidence from the emission probabilities. Finally, it resamples from the normalized weights to redistribute the precious particle resources.
- Compared to the forward-backward algorithm, both beam search and particle filtering can scale up to a large number of locations (assuming most of them are unlikely). Unlike beam search, however, particle filtering uses randomness to ensure better diversity of the particles.
- Particle filtering is also called sequential Monte Carlo and there are many more sophisticated extensions that I'd encourage to learn about.

# Other inference methods: Gibbs sampling

**Algorithm: Gibbs sampling**

Initialize $x$ to a random complete assignment

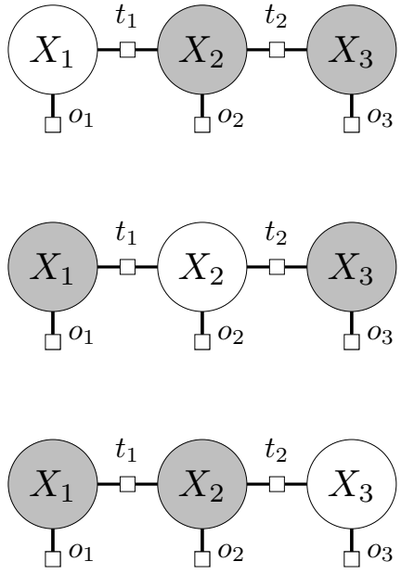
Loop through $i = 1, \ldots, n$ until convergence:

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

($X_{-i}$ denotes all variables except $X_i$)

Increment $\text{count}_i(x_i)$

Estimate $\hat{\mathbb{P}}(X_i = x_i) = \frac{\text{count}_i(x_i)}{\sum_v \text{count}_i(v)}$

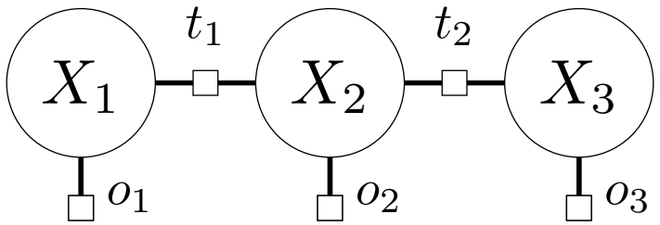



**Example: sampling one variable**

| | |
|---|---|
| $\text{Weight}(x \cup \{X_2 : 0\}) = 1$ | prob. $0.2$ |
| $\text{Weight}(x \cup \{X_2 : 1\}) = 2$ | prob. $0.4$ |
| $\text{Weight}(x \cup \{X_2 : 2\}) = 2$ | prob. $0.4$ |


[demo]

# Summary




| Algorithm | Strategy | Optimality | Time complexity |
|---|---|---|---|
| Forward backward | dynamic programming | exact | quadratic |
| Particle Filter | extend partial assignments | approximate | linear |
| Gibbs Sampling | modify complete assignments | approximate | linear |