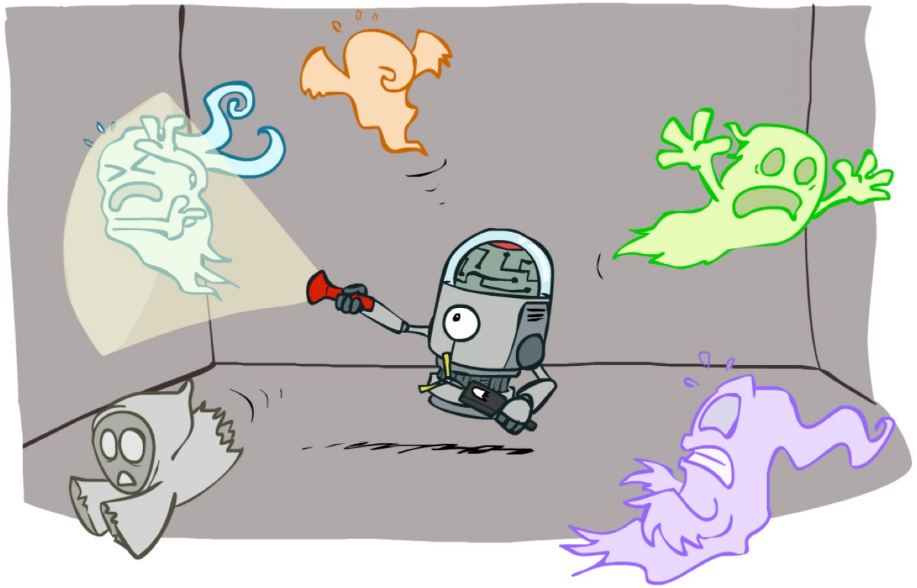
Artificial Intelligence - INFOF311

HMM, particle filters and DBN

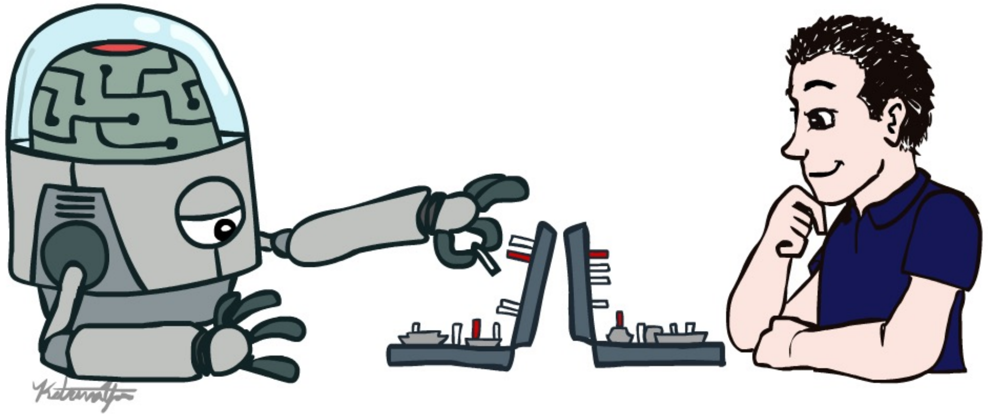


Instructor : Tom Lenaerts

Acknowledgement

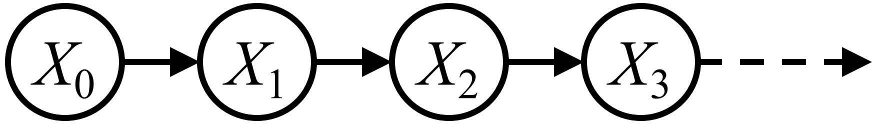
We thank Stuart Russell for his generosity in allowing us to use the slide set of the UC Berkeley Course CS188, Introduction to Artificial Intelligence. These slides were created by Dan Klein, Pieter Abbeel and Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.

CHAI Center for Human-Compatible Artificial Intelligence



The slides for INFOF311 are slightly modified versions of the slides of the spring and summer CS188 sessions in 2021 and 2022

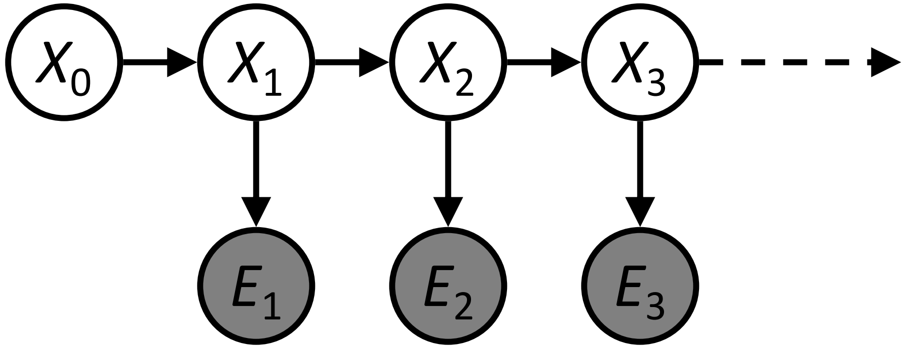
Markov Chains



$P(X_0)$ $\quad$ $P(X_t \mid X_{t-1})$

- ***Stationarity*** assumption: transition probabilities are the same at all times
- ***Markov*** assumption: "future is independent of the past given the present"
- Mini-Forward Algorithm: $P(X_t) = \sum_{x_{t-1}} P(X_{t-1}=x_{t-1})\, P(X_t \mid X_{t-1}=x_{t-1})$
- Equivalently: $P_{t+1} = T^{\mathrm{T}} P_t$
- Stationary Distribution: $P_\infty = T^{\mathrm{T}} P_\infty$
- Stationary distribution does not depend on the starting distribution

Hidden Markov Models



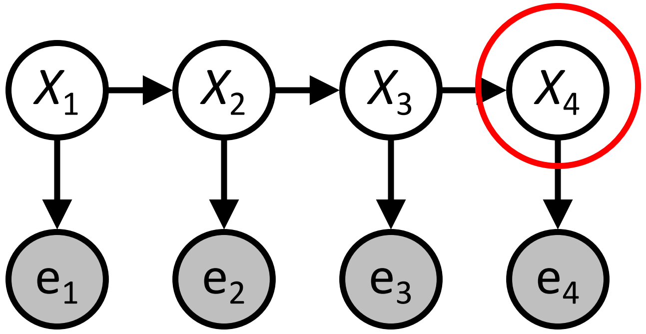
- Sensor models are the same at all times
- Current evidence is independent of everything else given the current state
- **Inference tasks**; Filtering, prediction, …

Inference tasks

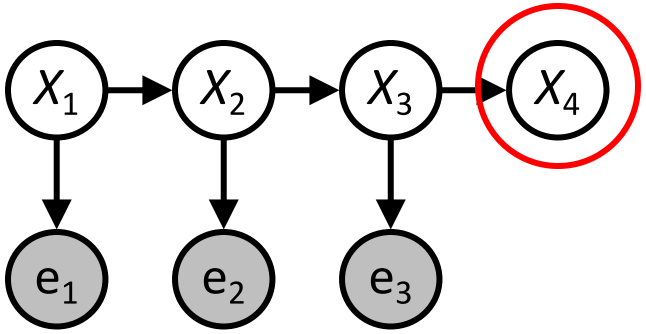
- ***Filtering***: $P(X_t | e_{1:t})$
  - ***belief state***—input to the decision process of a rational agent
- ***Prediction***: $P(X_{t+k} | e_{1:t})$ for $k > 0$
  - evaluation of possible action sequences; like filtering without the evidence
- ***Smoothing***: $P(X_k | e_{1:t})$ for $0 \leq k < t$
  - better estimate of past states, essential for learning
- ***Most likely explanation***: $\arg\max_{x_{1:t}} P(x_{1:t} \mid e_{1:t})$
  - speech recognition, decoding with a noisy channel

Inference tasks

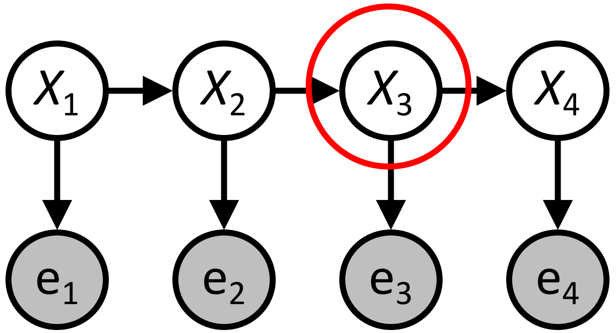
Filtering: $P(X_t | e_{1:t})$



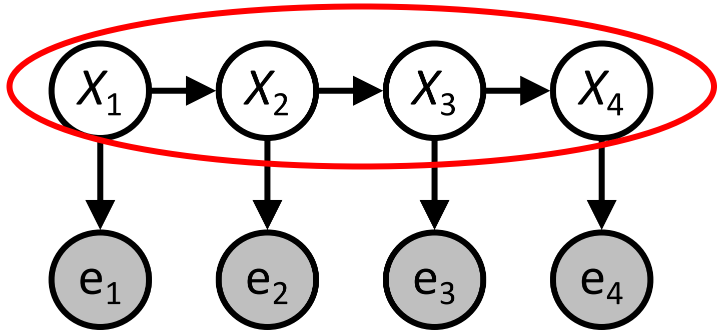
Prediction: $P(X_{t+k} | e_{1:t})$



Smoothing: $P(X_k | e_{1:t})$, $k<t$



Explanation: $P(X_{1:t} | e_{1:t})$



Filtering / Monitoring

- Filtering, or monitoring, or state estimation, is the task of maintaining the distribution $\boldsymbol{f}_{1:t} = P(X_t | e_{1:t})$ over time

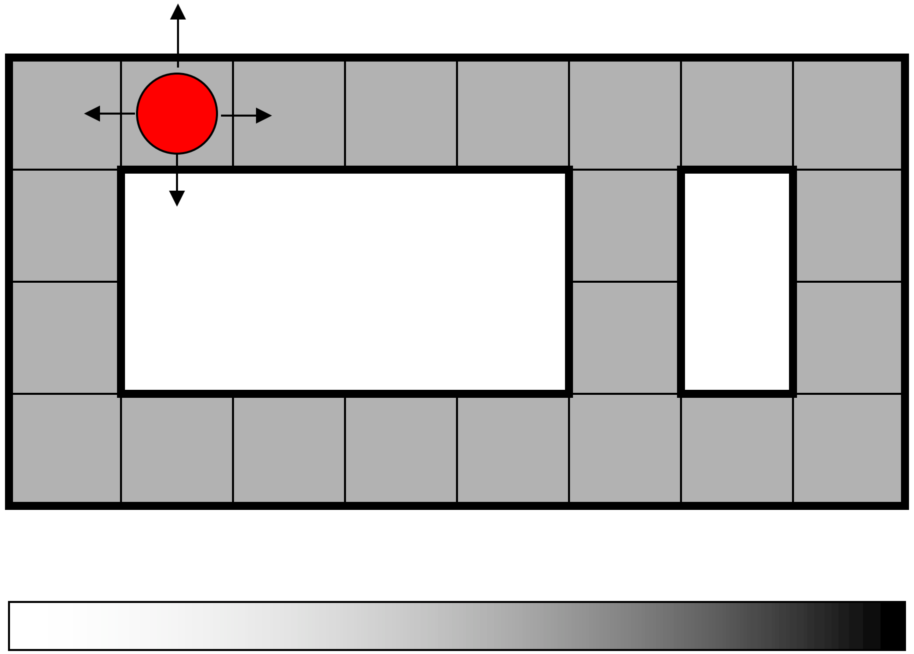
- We start with $\boldsymbol{f}_0$ in an initial setting, usually uniform

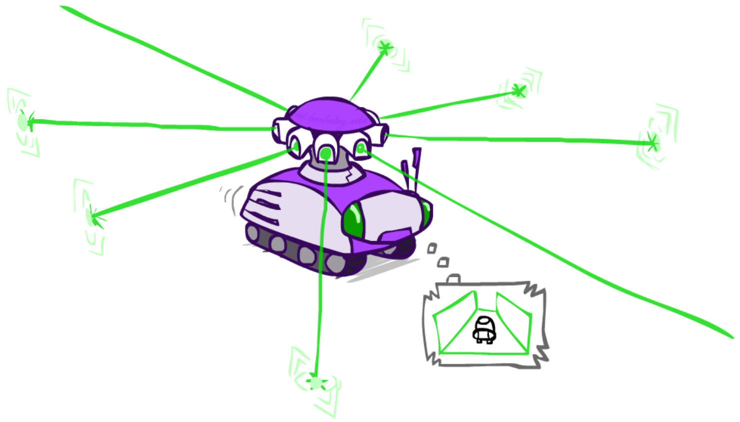
- Filtering is a fundamental task in engineering and science

- The Kalman filter (continuous variables, linear dynamics, Gaussian noise) was invented in 1960 and used for trajectory estimation in the Apollo program; core ideas used by Gauss for planetary observations; 1110000 papers on Google Scholar (spring 2023).

Example: Robot Localization

Example from Michael Pfeiffer





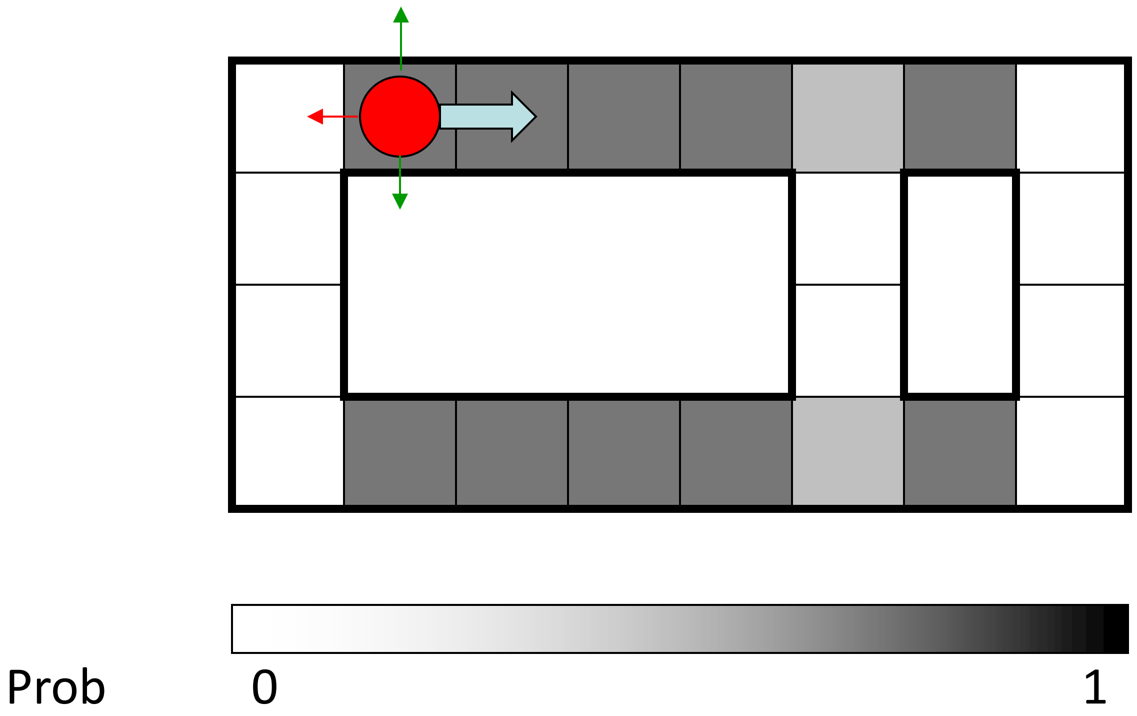
Prob 0 1

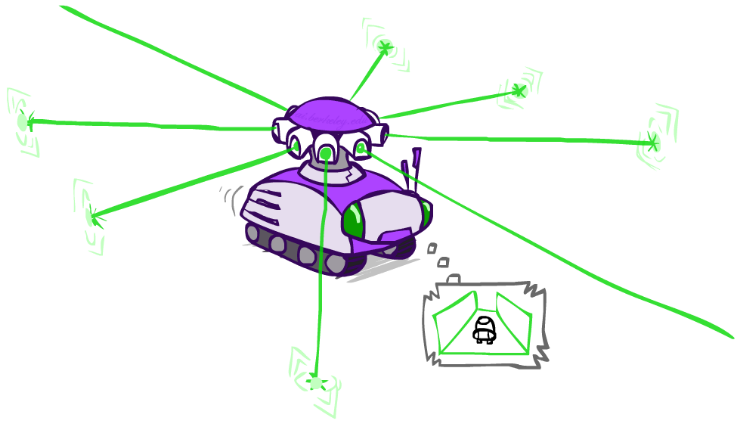
t=0

Sensor model: four bits for wall/no-wall in each direction, never more than 1 mistake

Transition model: action may fail with small prob.

Example: Robot Localization



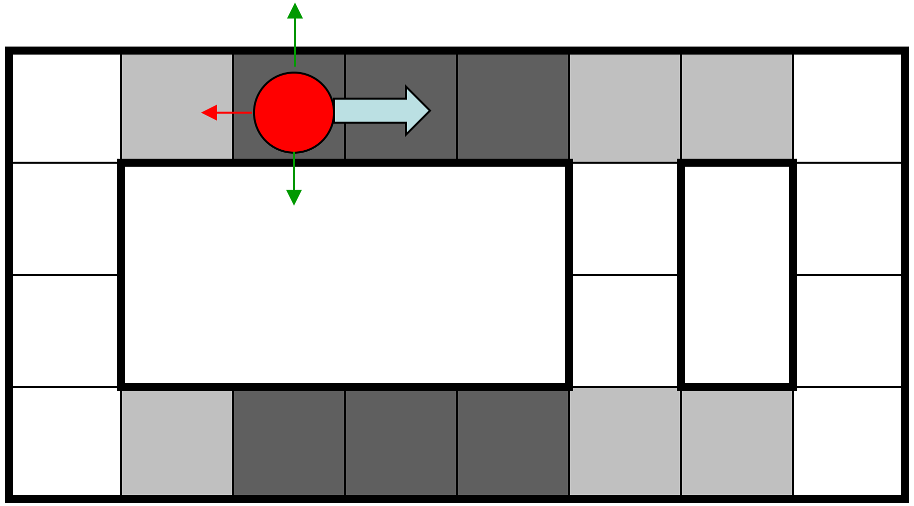


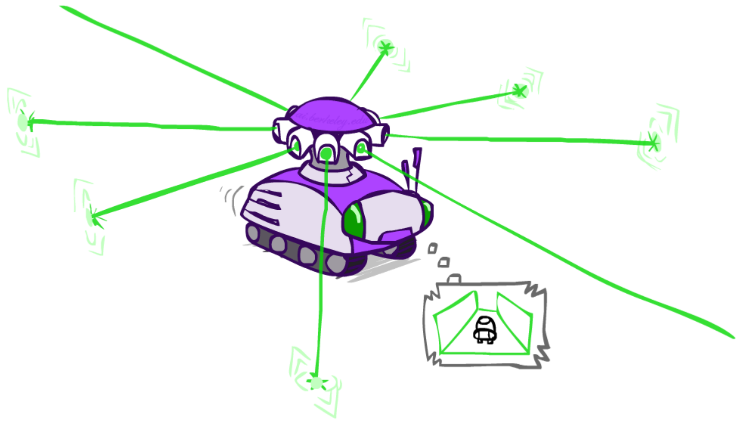
t=1

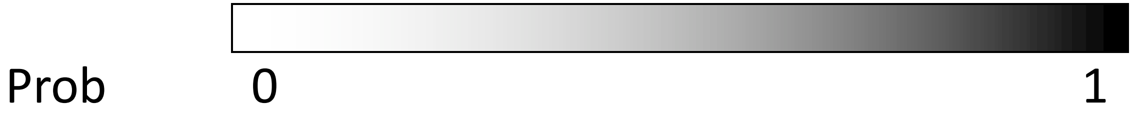
Lighter grey: was ***possible*** to get the reading,

but ***less likely*** (required 1 mistake)

Example: Robot Localization

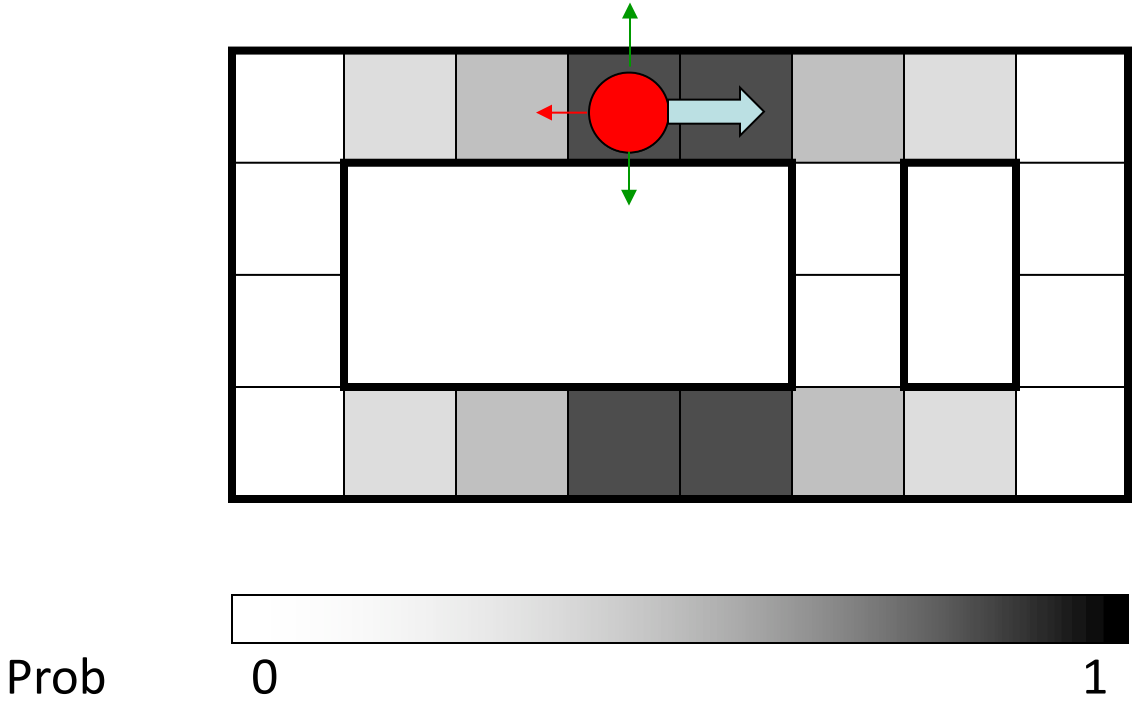


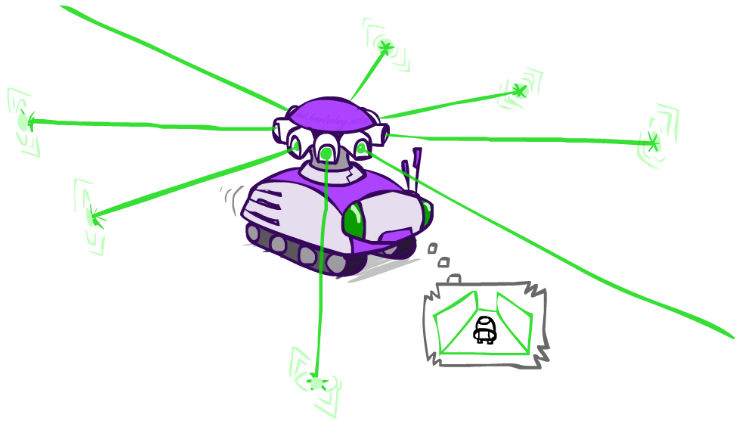




t=2

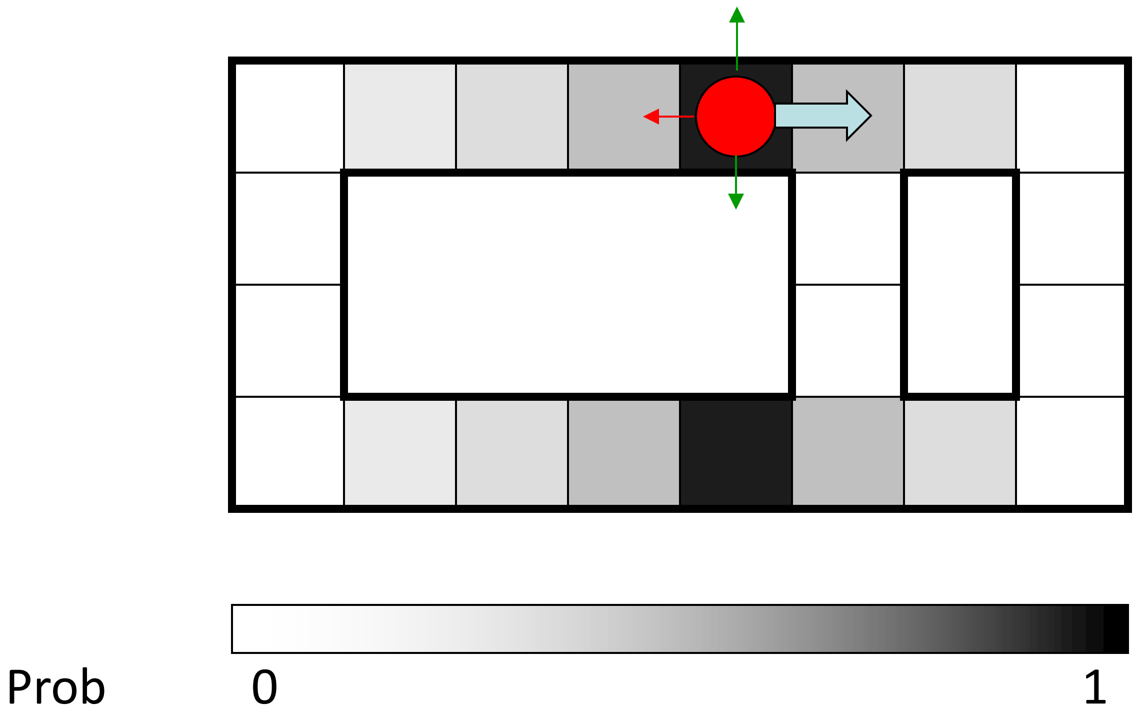
Example: Robot Localization

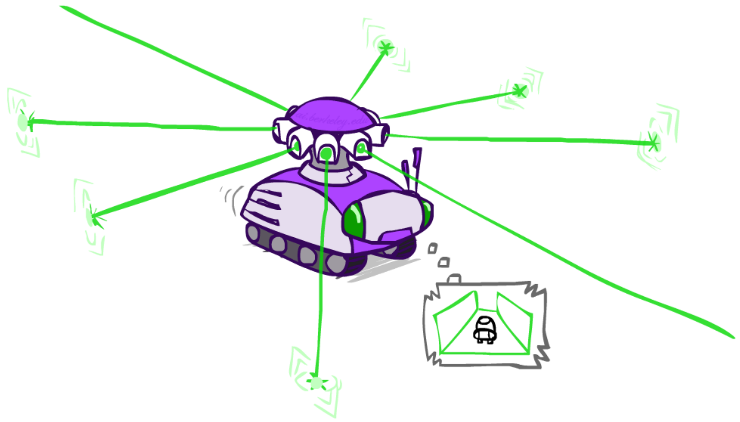




t=3

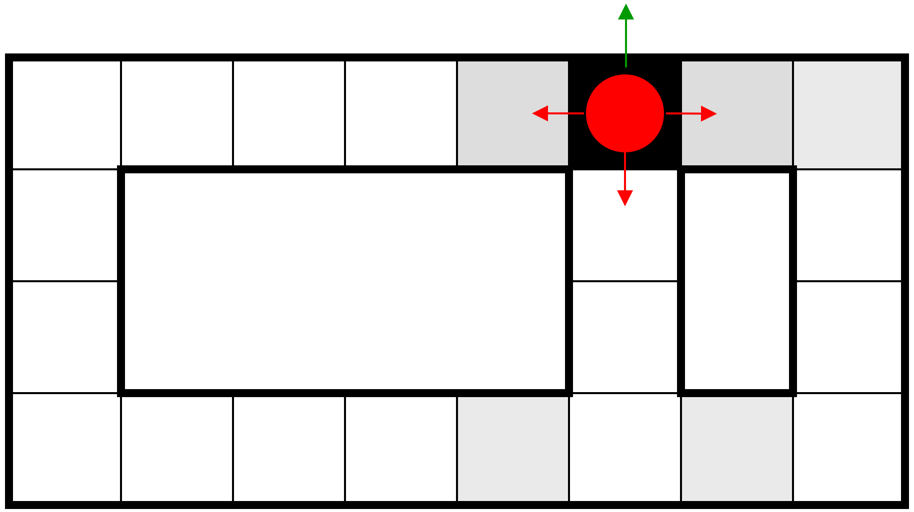
Example: Robot Localization

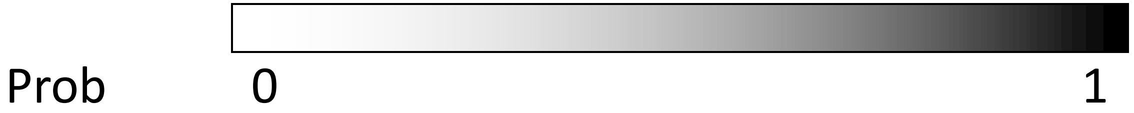




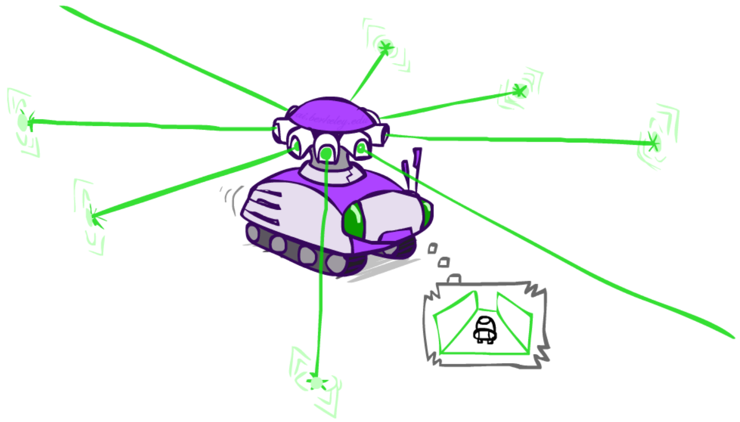
t=4

Example: Robot Localization





t=5



Filtering algorithm

- Aim: devise a ***recursive filtering*** algorithm of the form
  - $P(X_{t+1} | e_{1:t+1}) = g(e_{t+1}, P(X_t | e_{1:t}))$

- $P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1})$
- $= \alpha\, P(e_{t+1} | X_{t+1}, e_{1:t})\, P(X_{t+1} | e_{1:t})$
- $= \alpha\, P(e_{t+1} | X_{t+1})\, P(X_{t+1} | e_{1:t})$
- $= \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t})\, P(X_{t+1} | x_t, e_{1:t})$
- $= \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t})\, P(X_{t+1} | x_t)$

Apply Bayes' rule

Apply conditional independence

Condition on X_t

Apply conditional independence

Normalize

Update

Predict

Filtering algorithm

- $P(X_{t+1} | e_{1:t+1}) = \underbrace{\alpha}_{\text{Normalize}} \underbrace{P(e_{t+1} | X_{t+1})}_{\text{Update}} \underbrace{\sum_{x_t} P(x_t \mid e_{1:t})\, P(X_{t+1} | x_t)}_{\text{Predict}}$

- $\boldsymbol{f}_{1:t+1}$ = FORWARD($\boldsymbol{f}_{1:t}$, e_{t+1})
- Cost per time step: $O(|X|^2)$ where $|X|$ is the number of states
- Time and space costs are ***constant***, independent of t
- $O(|X|^2)$ is infeasible for models with many state variables
- We get to invent really cool approximate filtering algorithms

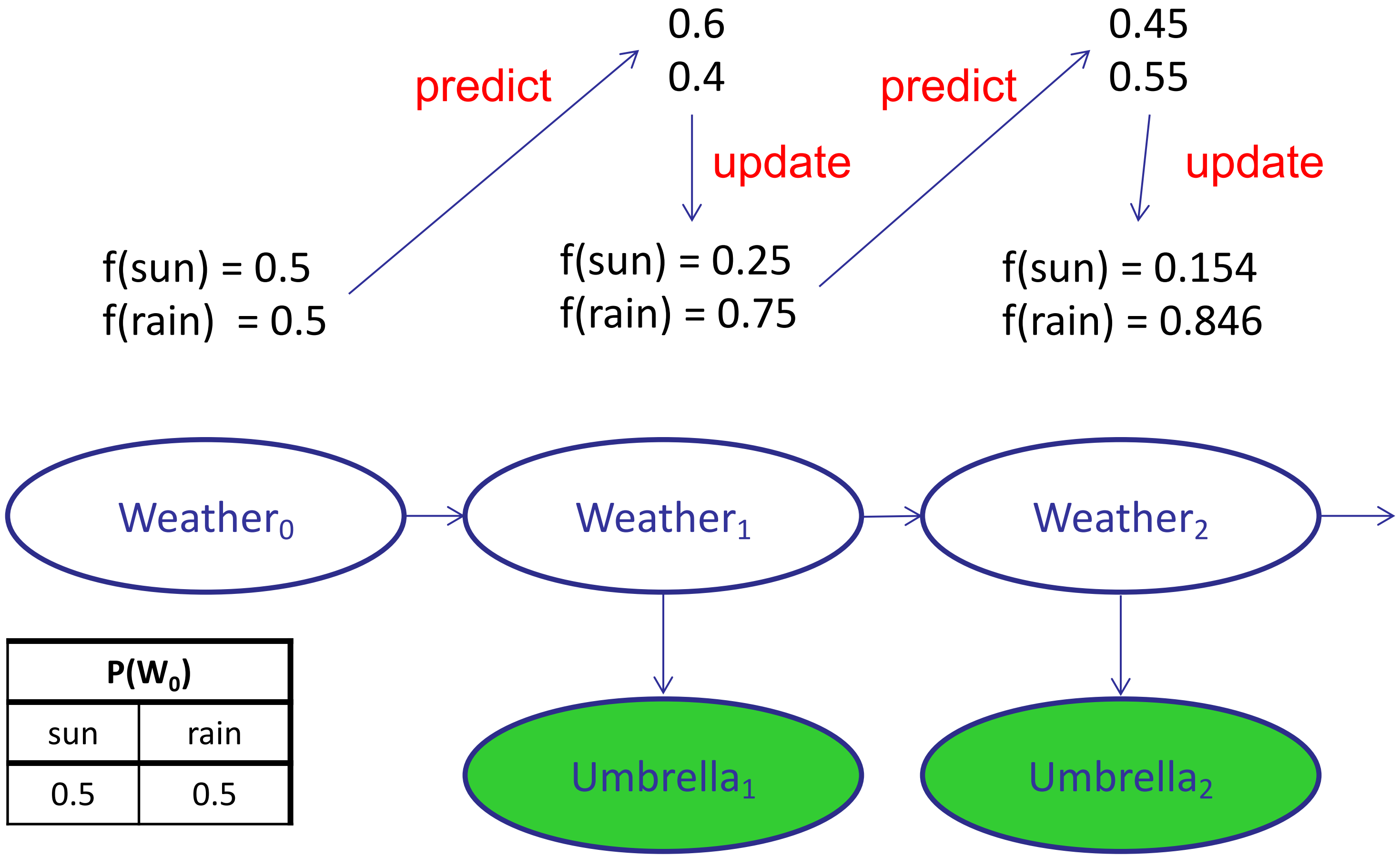
And the same thing in linear algebra

- Transition matrix T, observation matrix O_t
  - Observation matrix has state likelihoods for E_t along diagonal
  - E.g., for U_1 = true, $O_1 = \begin{pmatrix} 0.2 & 0 \\ 0 & 0.9 \end{pmatrix}$
- Filtering algorithm becomes
  - $\boldsymbol{f}_{1:t+1} = \alpha\, O_{t+1} T^{\mathrm{T}}\, \boldsymbol{f}_{1:t}$

X_{t-1}	$P(X_t \mid X_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

W_t	$P(U_t \mid W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

Example: Weather HMM

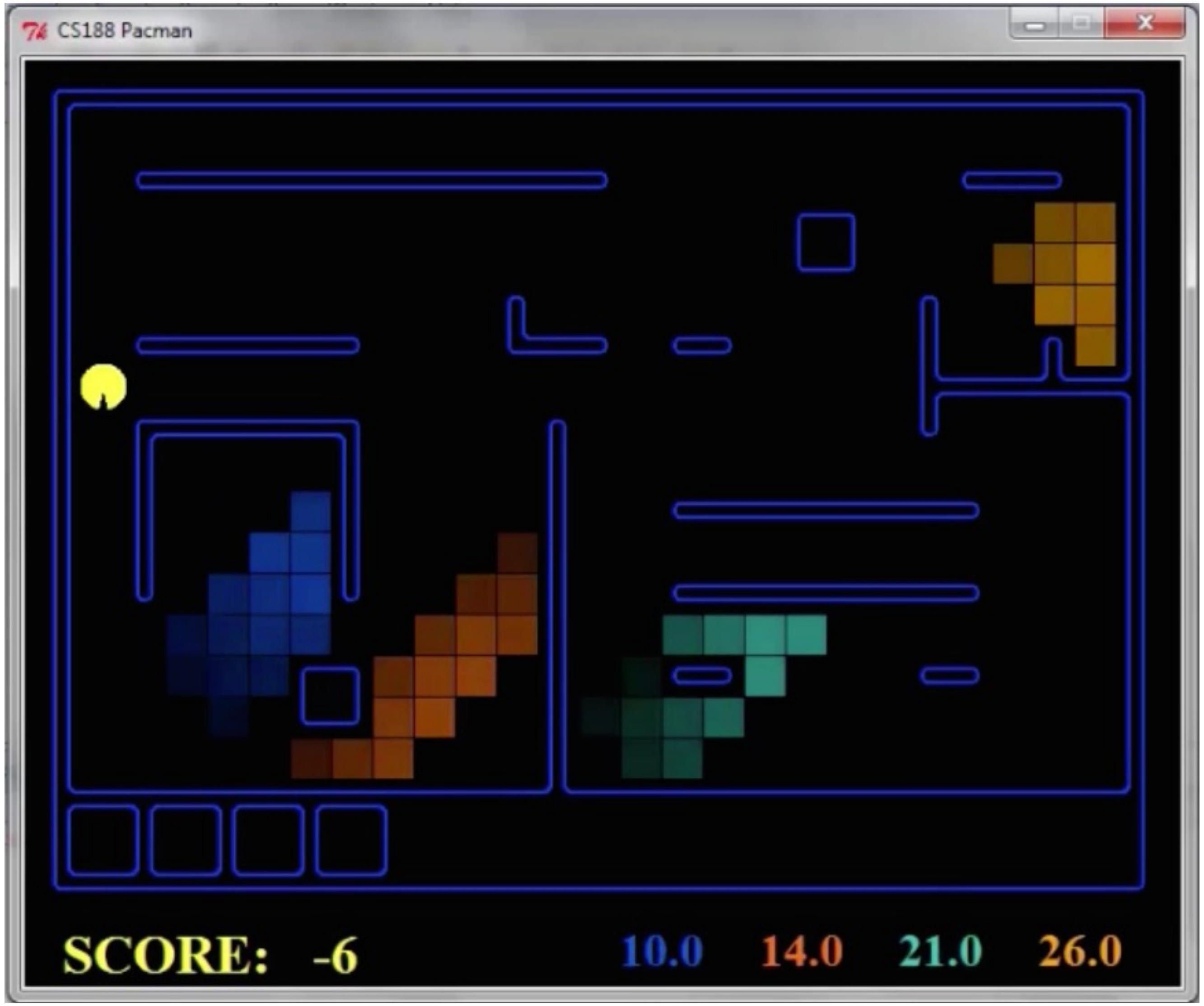


P(W_0)	
sun	rain
0.5	0.5

W_{t-1}	P(W_t|W_{t-1})	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

W_t	P(U_t|W_t)	
	true	false
sun	0.2	0.8
rain	0.9	0.1

Pacman – Hunting Invisible Ghosts with Sonar



[Demo: Pacman – Sonar – No Beliefs(L14D1)]

Video of Demo Pacman – Sonar (with beliefs)

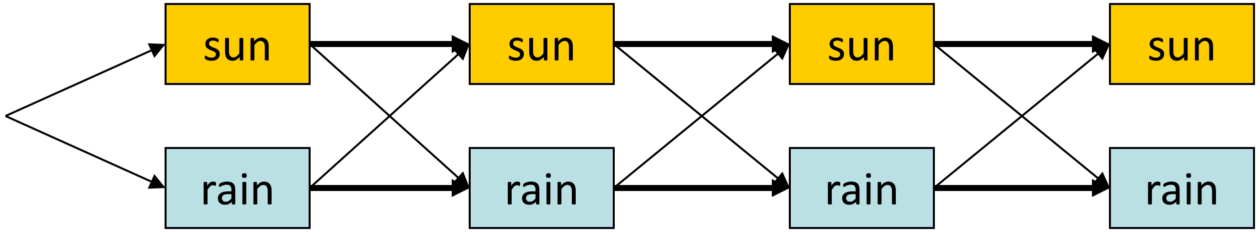
Most Likely Explanation

Inference tasks

- ***Filtering***: $P(X_t | e_{1:t})$
  - ***belief state***—input to the decision process of a rational agent
- ***Prediction***: $P(X_{t+k} | e_{1:t})$ for $k > 0$
  - evaluation of possible action sequences; like filtering without the evidence
- ***Smoothing***: $P(X_k | e_{1:t})$ for $0 \leq k < t$
  - better estimate of past states, essential for learning
- ***Most likely explanation***: $\arg\max_{x_{1:t}} P(x_{1:t} \mid e_{1:t})$
  - speech recognition, decoding with a noisy channel

Most likely explanation = most probable path

- ***State trellis***: graph of states and transitions over time

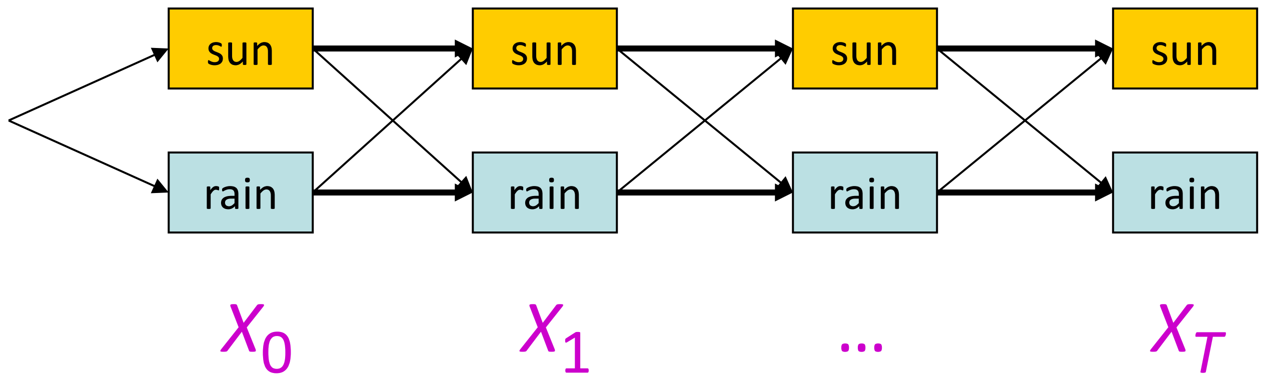


X_0 X_1 … X_T

- $\arg\max_{x_{1:t}} P(x_{1:t} \mid e_{1:t})$
  $= \arg\max_{x_{1:t}} \alpha\, P(x_{1:t}, e_{1:t})$
  $= \arg\max_{x_{1:t}} P(x_{1:t}, e_{1:t})$
  $= \arg\max_{x_{1:t}} P(x_0) \prod_t P(x_t \mid x_{t-1}) P(e_t \mid x_t)$

- Each arc represents some transition $x_{t-1} \rightarrow x_t$
- Each arc has weight $P(x_t \mid x_{t-1})\, P(e_t \mid x_t)$ (arcs to initial states have weight $P(x_0)$)
- The ***product*** of weights on a path is proportional to that state sequence's probability
- Forward algorithm computes sums of paths, ***Viterbi algorithm*** computes best paths

Forward / Viterbi algorithms



Forward Algorithm (sum)

For each state at time t, keep track of the ***total probability of all paths*** to it

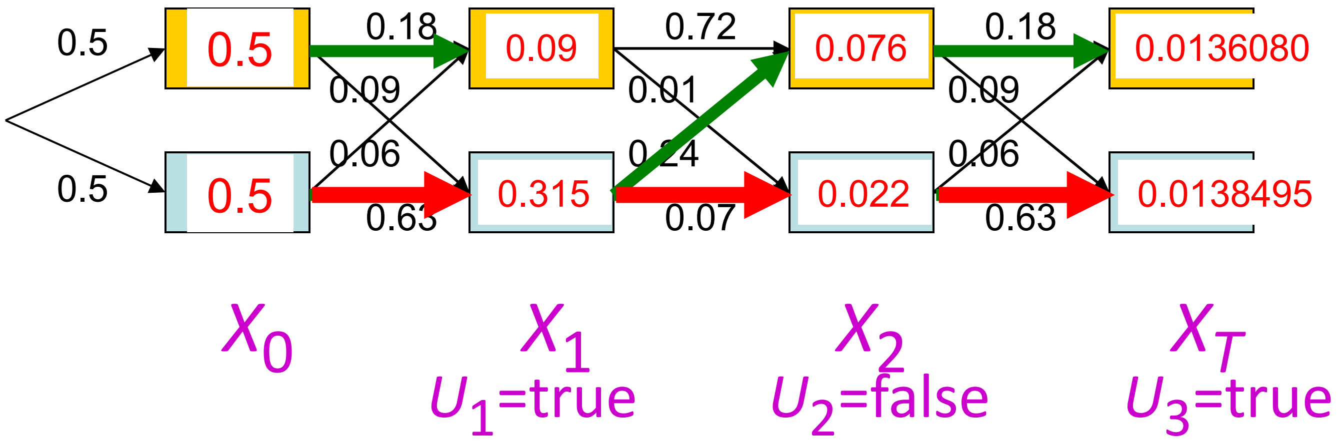
$$\mathbf{f}_{1:t+1} = \text{FORWARD}(\mathbf{f}_{1:t}, e_{t+1})$$
$$= \alpha\, P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|\, x_t)\, \mathbf{f}_{1:t}$$

Viterbi Algorithm (max)

For each state at time t, keep track of the ***maximum probability of any path*** to it

$$\mathbf{m}_{1:t+1} = \text{VITERBI}(\mathbf{m}_{1:t}, e_{t+1})$$
$$= P(e_{t+1}|X_{t+1}) \max_{x_t} P(X_{t+1}|\, x_t)\, \mathbf{m}_{1:t}$$

Viterbi algorithm contd.



W_{t-1}	$P(W_t \mid W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

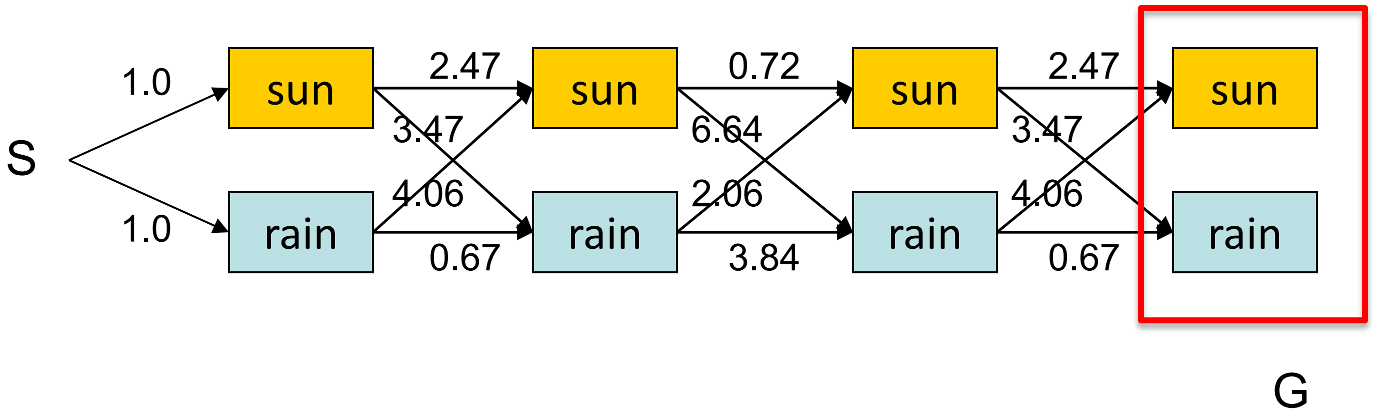
W_t	$P(U_t \mid W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

Time complexity?
$O(|X|^2 T)$

Space complexity?
$O(|X| T)$

Number of paths?
$O(|X|^T)$

Viterbi in negative log space



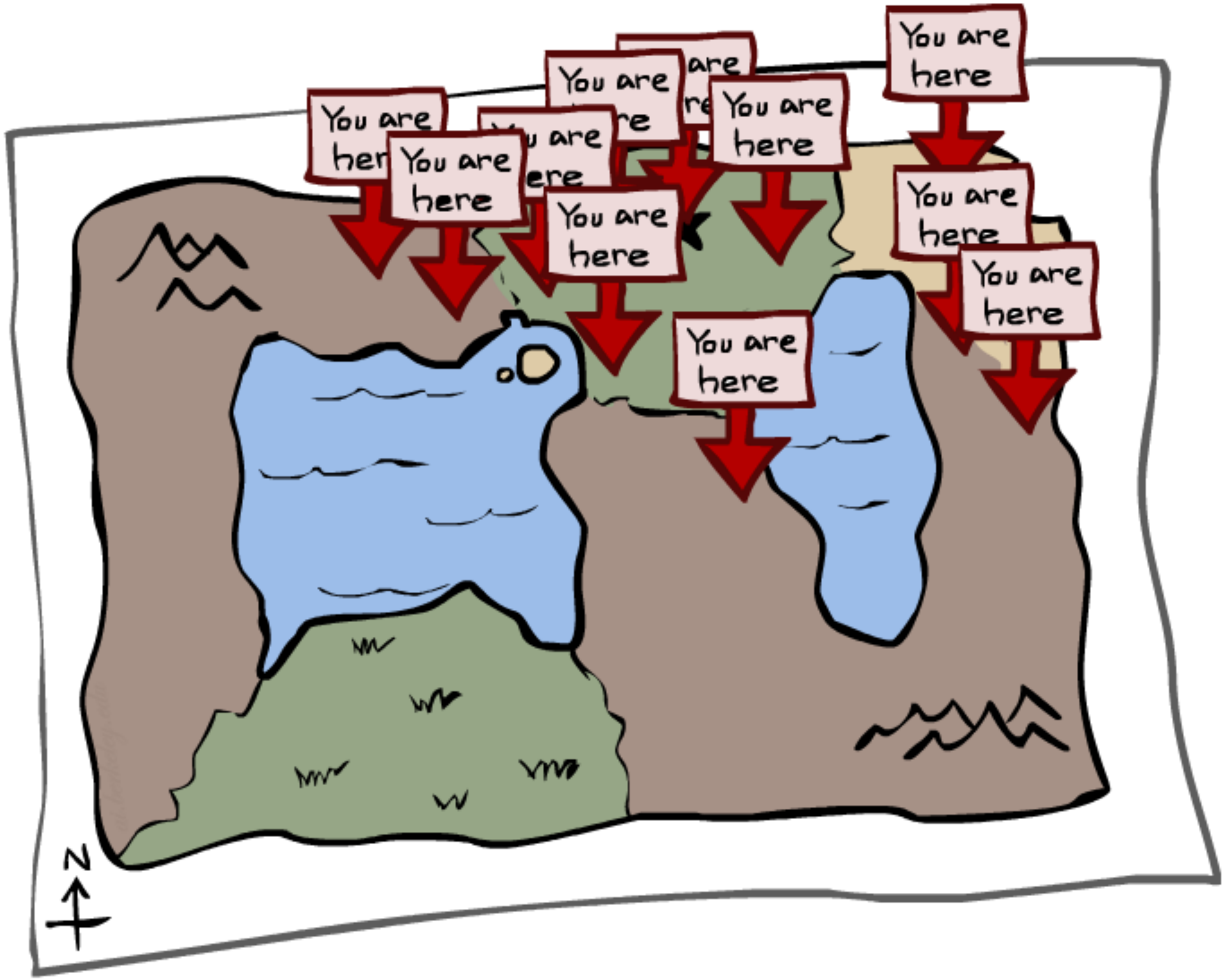
W_{t-1}	$P(W_t \mid W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

W_t	$P(U_t \mid W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

argmax of product of probabilities
= argmin of sum of negative log probabilities
= minimum-cost path

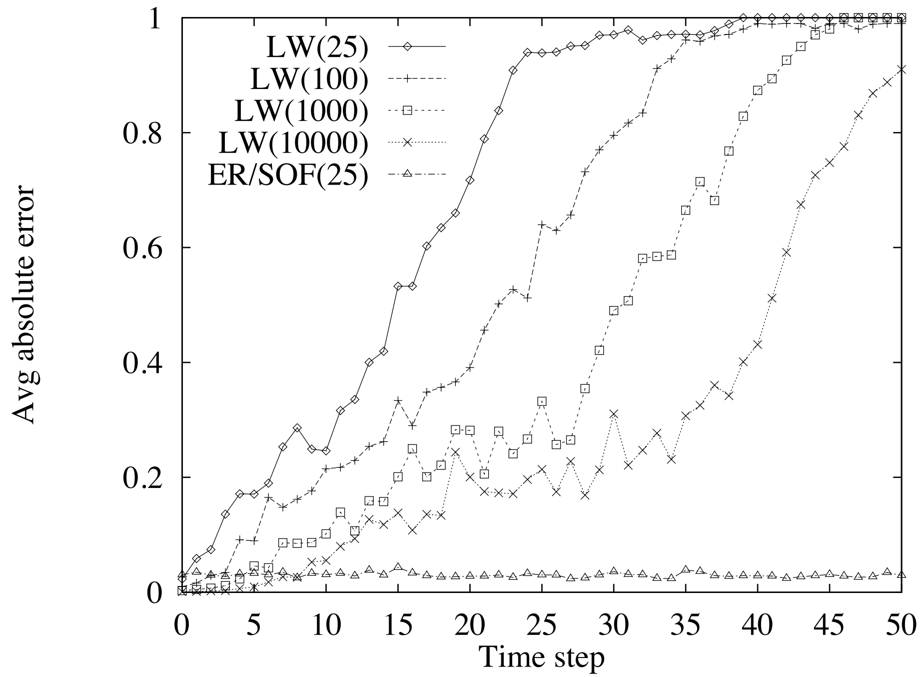
Viterbi is essentially breadth-first graph search
What about A*?

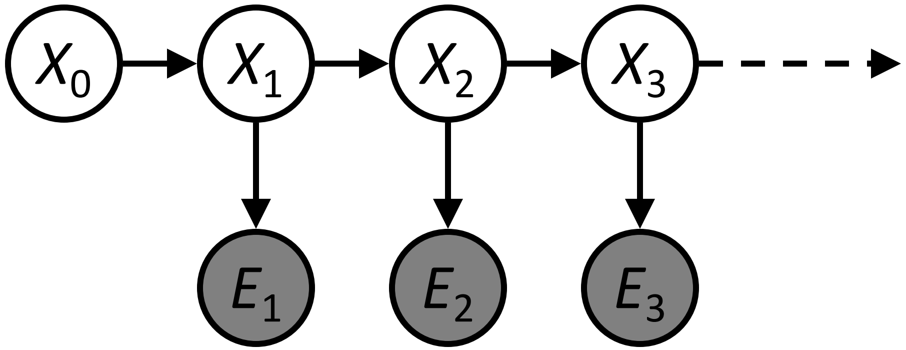
Particle Filtering



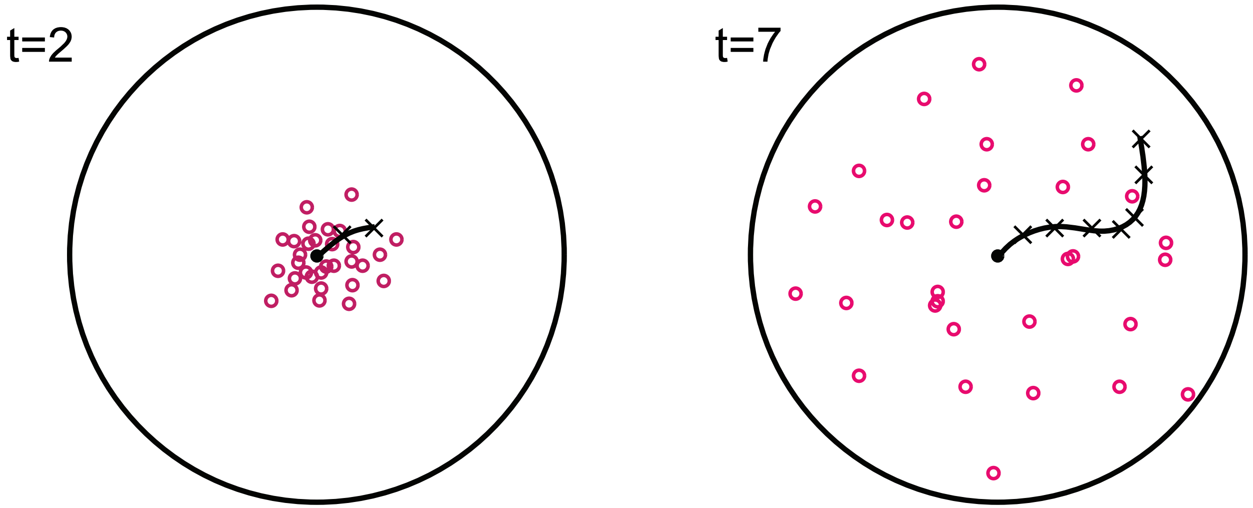
We need a new algorithm!

- When size of the state space is large, exact inference becomes infeasible
- Likelihood weighting also fails completely – number of samples needed grows ***exponentially*** with *T*





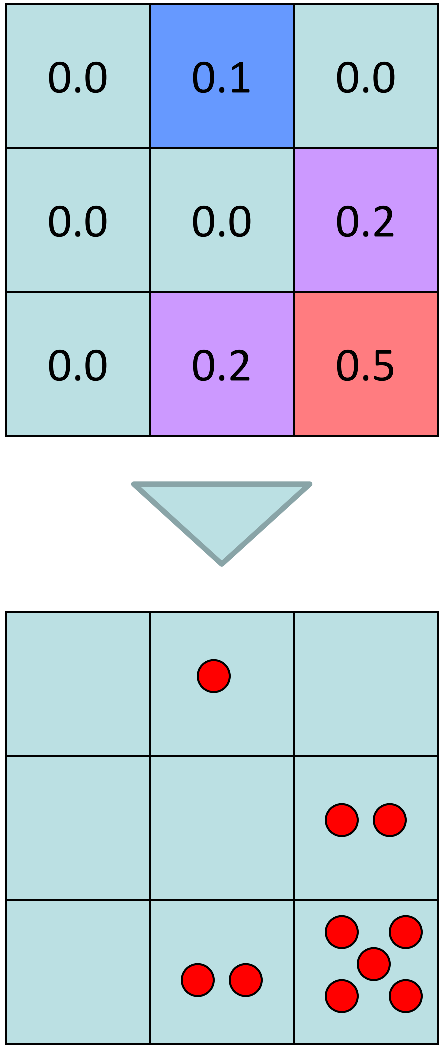
Particle Filtering: Resample



- The problem of likelihood weighting: sample state trajectories go off into low-probability regions; too few “reasonable” samples
- Solution: kill the bad ones, make more of the good ones
- This way the population of samples stays in the high-probability region

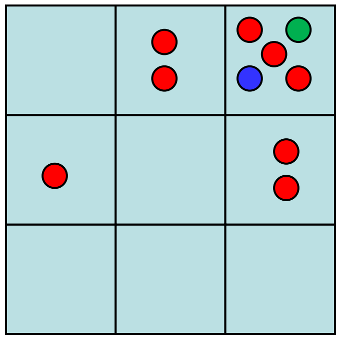
Particle Filtering

- Solution: approximate inference
  - Track samples of X, not all values
  - Samples are called particles
  - Time per step is linear in the number of samples
  - But: number needed may be large
  - In memory: list of particles, not states



Representation: Particles

- Our representation of $P(X)$ is now a list of $N << |X|$ particles
- $P(x)$ approximated by number of particles with value x
  - So, many x may have $P(x) = 0$!
  - More particles => more accuracy
- For now, all particles have a weight of 1



Particles:
(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,2)
(1,2)
(3,3)
(3,3)
(2,3)

Particle Filtering: Prediction Step

- Each particle is moved by sampling its next position from the transition model

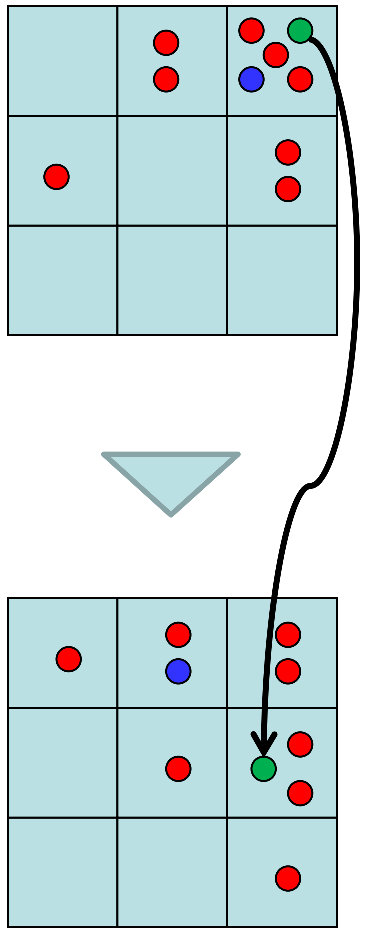
$$x' = \text{sample}(P(X'|x))$$

  - This is like prior sampling – samples' frequencies reflect the transition probabilities
  - Here, most samples move clockwise, but some move in another direction or stay in place

- This captures the passage of time
  - If enough samples, close to exact values before and after (consistent)

Particles:
(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,2)
(1,2)
(3,3)
(3,3)
(2,3)

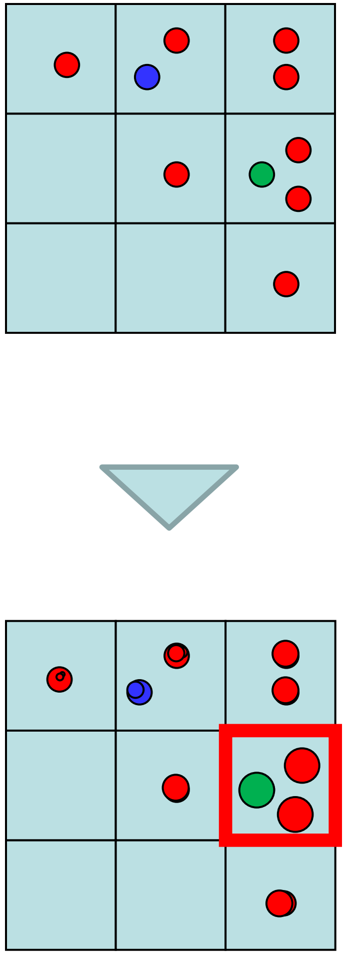
Particles:
(3,2)
(2,3)
(3,2)
(3,1)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(2,2)



Particle Filtering: Update step

- After observing e_{t+1} :
  - As in likelihood weighting, weight each sample based on the evidence
    - $w^{(j)} = P(e_{t+1} \mid x_{t+1}^{(j)})$
  - Normalize the weights: particles that fit the data better get higher weights, others get lower weights

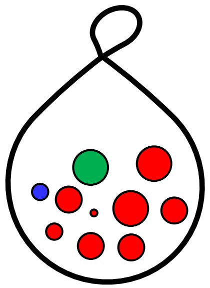
Particles:
(3,2)
(2,3)
(3,2)
(3,1)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(2,2)



Particles:
(3,2) w=.9
(2,3) w=.2
(3,2) w=.9
(3,1) w=.4
(3,3) w=.4
(3,2) w=.9
(1,3) w=.1
(2,3) w=.2
(3,2) w=.9
(2,2) w=.4

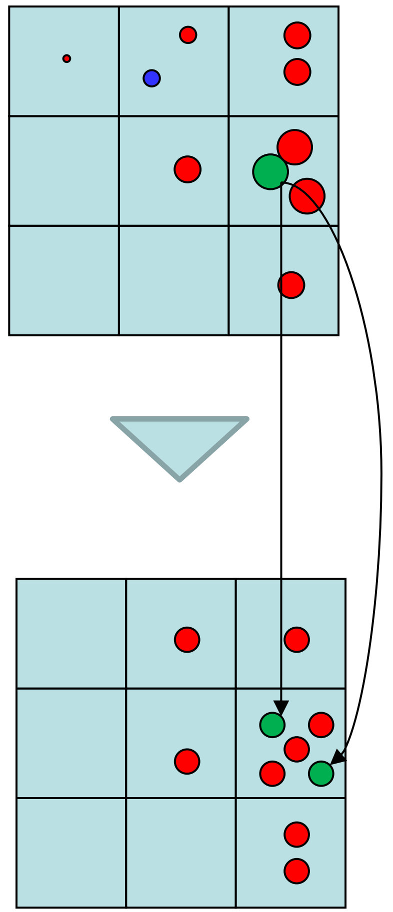
Particle Filtering: Resample

- Rather than tracking weighted samples, we ***resample***
- *N* times, we choose from our weighted sample distribution (i.e., draw with replacement)
- Now the update is complete for this time step, continue with the next one (with weights reset to 1)



Particles:
(3,2) w=.9
(2,3) w=.2
(3,2) w=.9
(3,1) w=.4
(3,3) w=.4
(3,2) w=.9
(1,3) w=.1
(2,3) w=.2
(3,2) w=.9
(2,2) w=.4

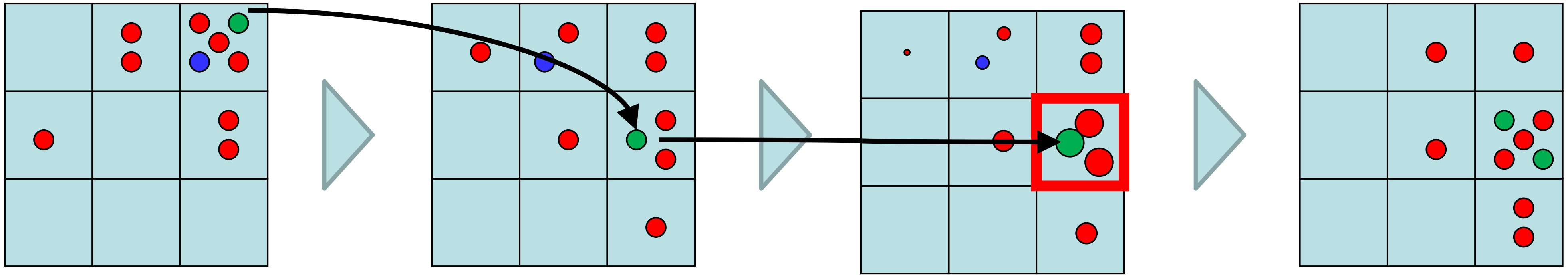
(New) Particles:
(3,2)
(2,2)
(3,2)
(2,3)
(3,3)
(3,2)
(3,1)
(3,1)
(3,2)
(3,2)



Summary: Particle Filtering

- Particles: track samples of states rather than an explicit distribution

Prediction Update/Weight Resample



Particles:	Particles:	Particles:	(New) Particles:
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,3)	(2,3) w=.2	(2,2)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(3,2)	(3,1)	(3,1) w=.4	(2,3)
(3,3)	(3,3)	(3,3) w=.4	(3,3)
(3,2)	(3,2)	(3,2) w=.9	(3,2)
(1,2)	(1,3)	(1,3) w=.1	(3,1)
(3,3)	(2,3)	(2,3) w=.2	(3,1)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,2)	(2,2) w=.4	(3,2)

Consistency: see proof in AIMA Ch. 14

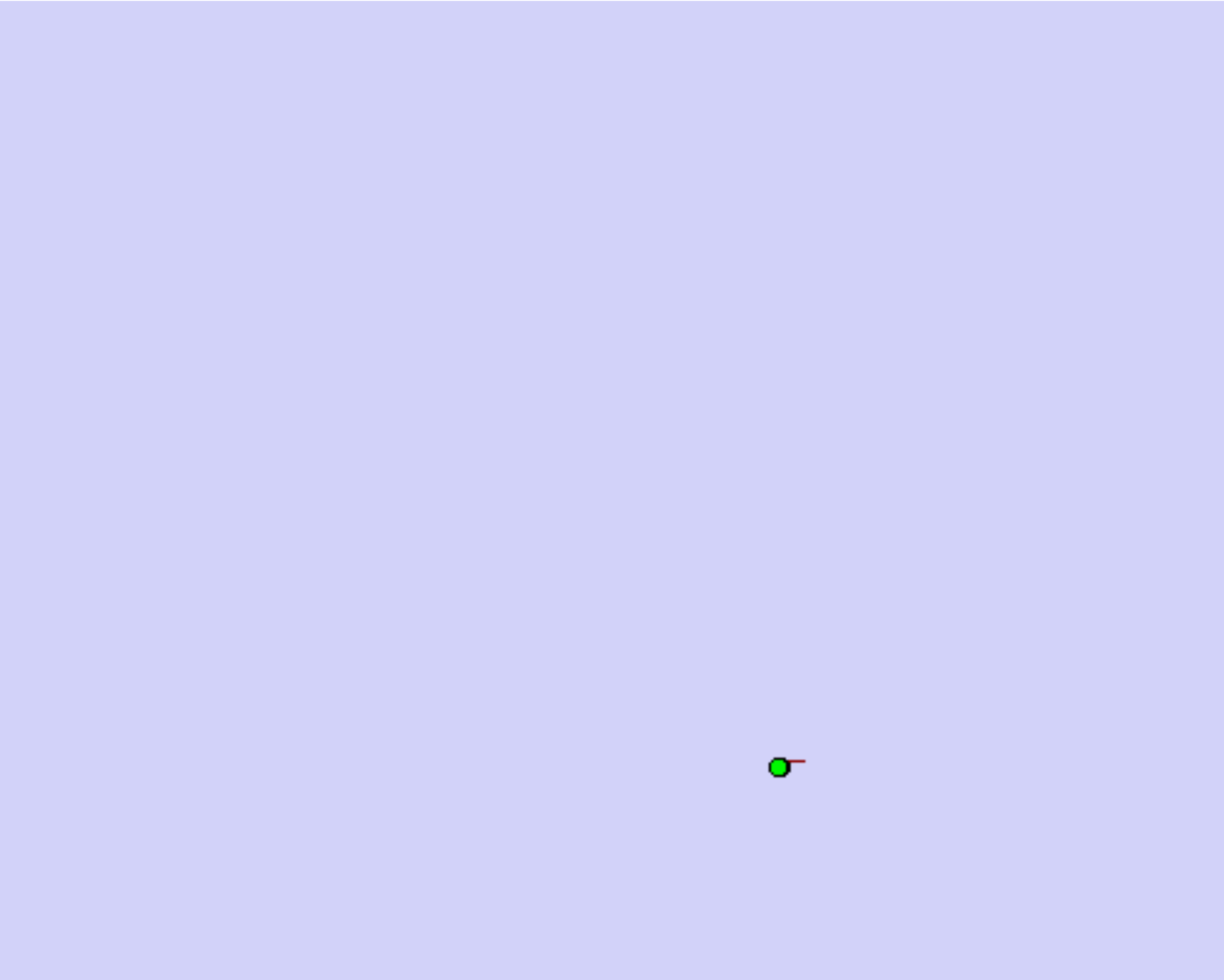
[Demos: ghostbusters particle filtering (L15D3,4,5)]

Particle Filter Localization (Sonar)

[Video: global-sonar-uw-annotated.avi]

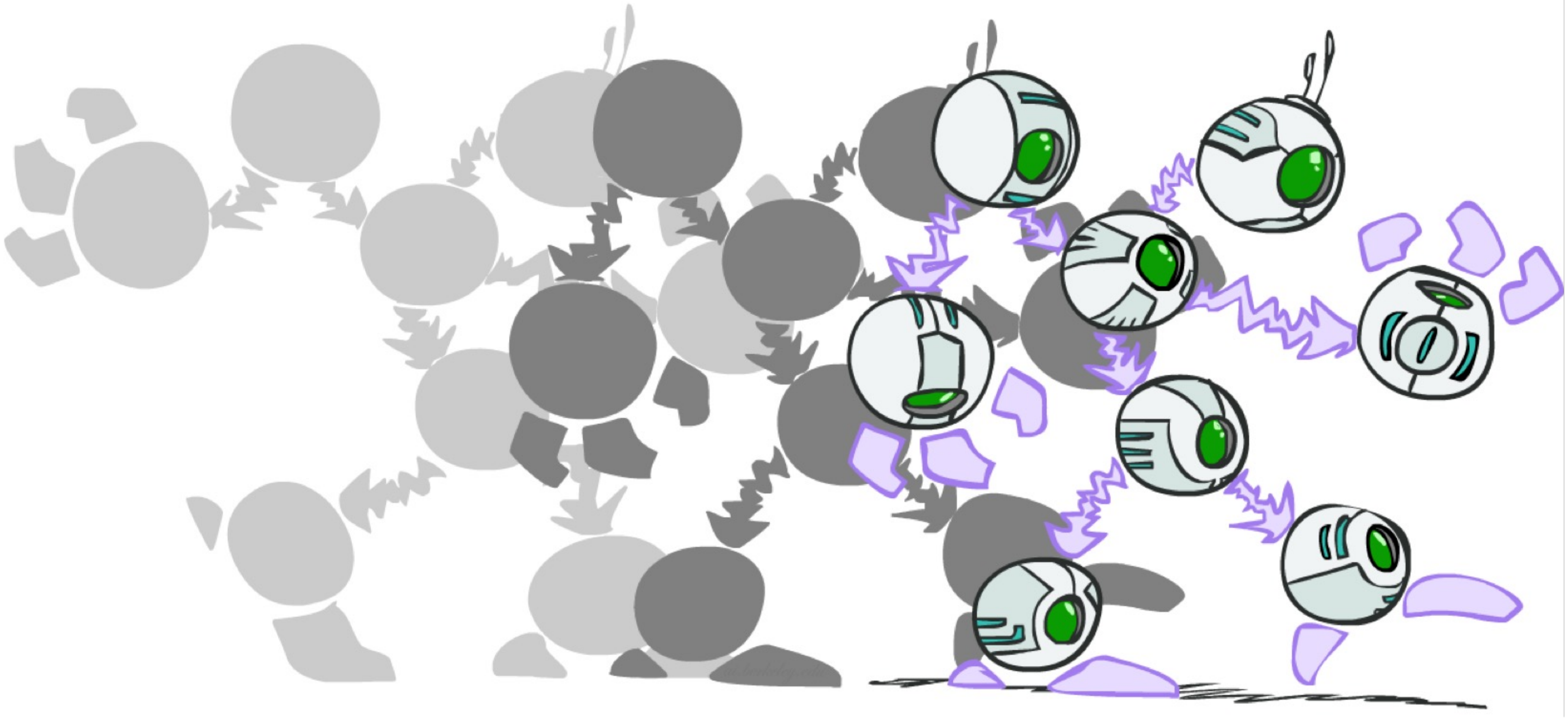
Particle Filter SLAM

Simultaneous localization and mapping



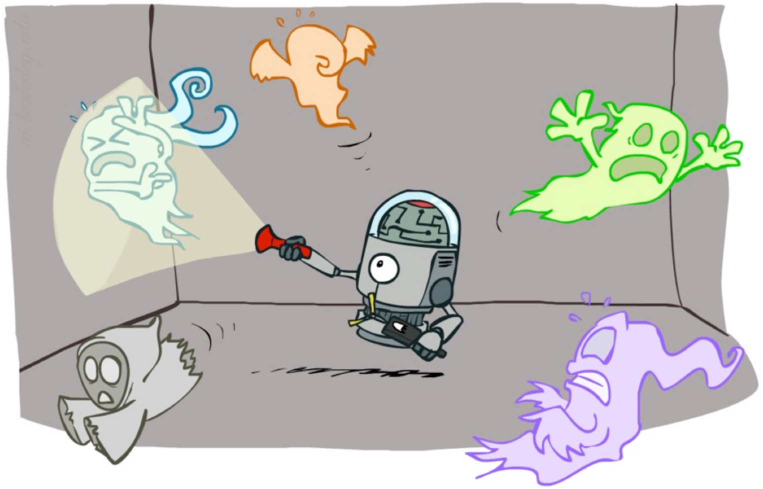
[Demo: PARTICLES-SLAM-fastslam.avi]

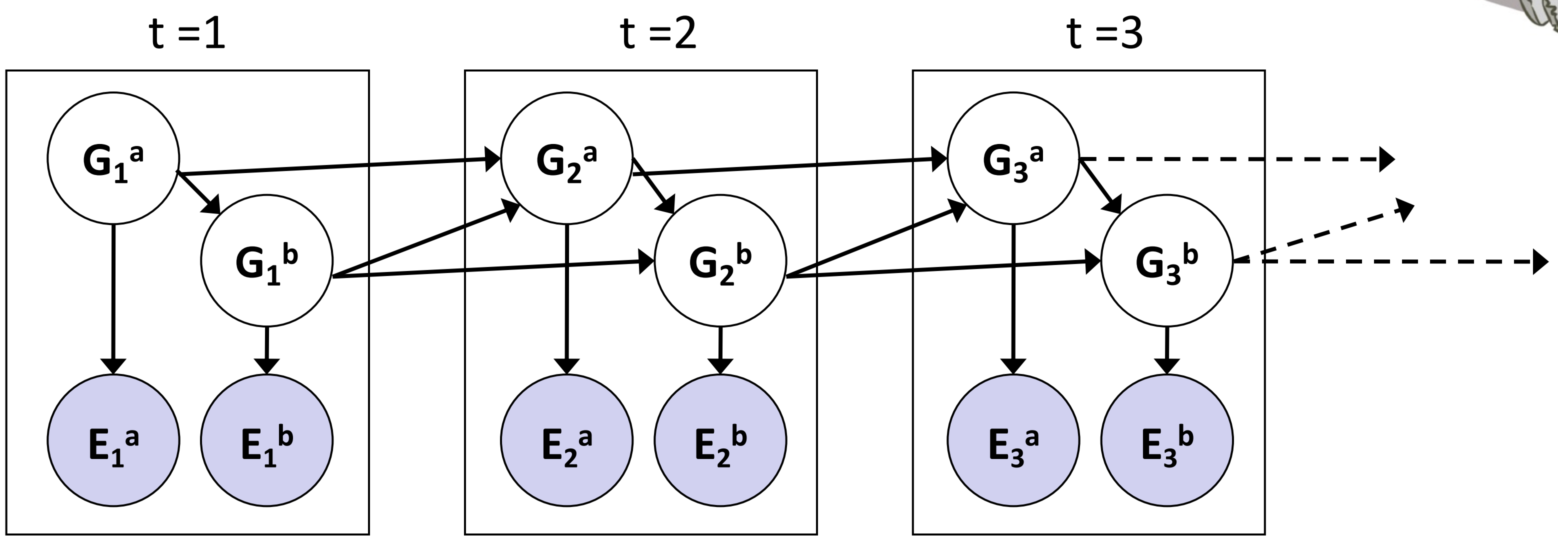
Dynamic Bayes Nets

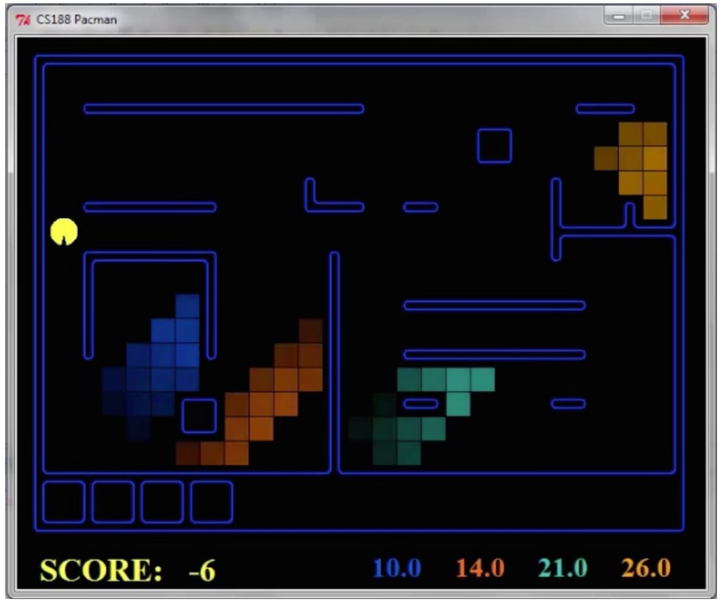


Dynamic Bayes Nets (DBNs)

- We want to track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net structure at each time
- Variables at time *t* can have parents at time *t-1*

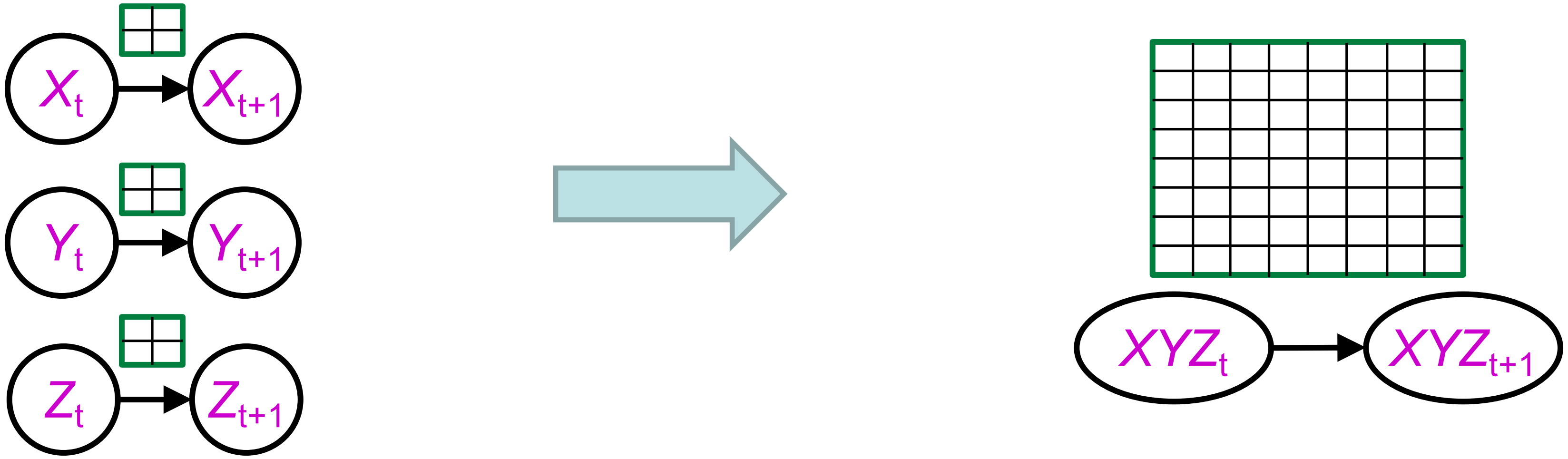






DBNs and HMMs

- Every HMM is a single-variable DBN
- Every discrete DBN is an HMM
  - HMM state is Cartesian product of DBN state variables

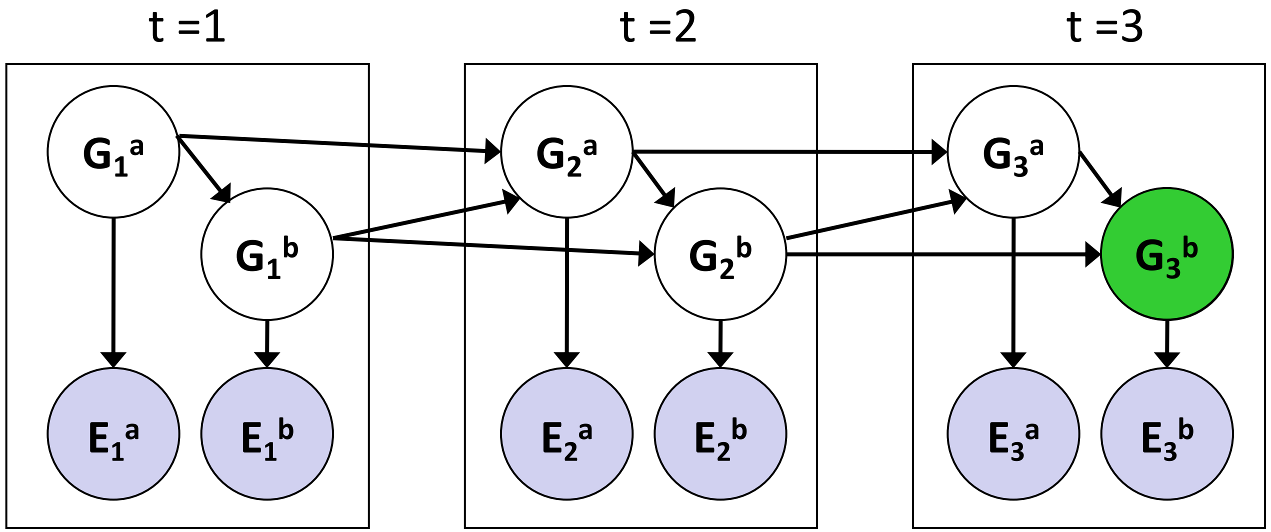


- Sparse dependencies => exponentially fewer parameters in DBN
  - E.g., 20 state variables, 3 parents each;

    DBN has $20 \times 2^3 = 160$ parameters, HMM has $2^{20} \times 2^{20} =\sim 10^{12}$ parameters

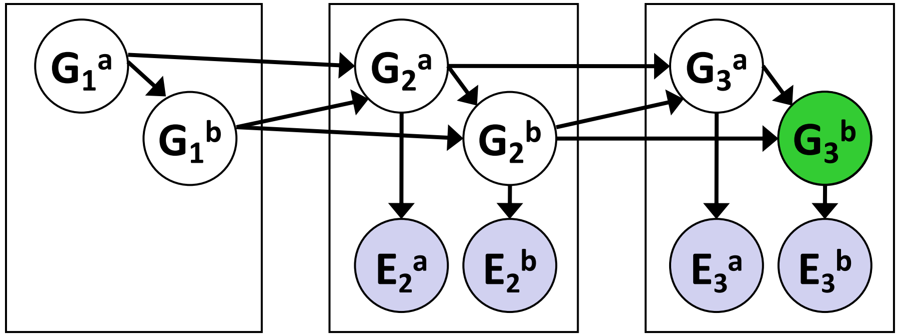
Exact Inference in DBNs

- Variable elimination applies to dynamic Bayes nets
- Offline: "unroll" the network for T time steps, then eliminate variables to find $P(X_T | e_{1:T})$



- Online: eliminate all variables from the previous time step; store factors for current time only
- Problem: largest factor contains all variables for current time (plus a few more)

DBN Particle Filters



- A particle is a complete sample for a time step
- **Initialize**: Generate prior samples for the t=1 Bayes net
  - Example particle: $\mathbf{G_1^a}$ = (3,3) $\mathbf{G_1^b}$ = (5,3)
- **Elapse time**: Sample a successor for each particle
  - Example successor: $\mathbf{G_2^a}$ = (2,3) $\mathbf{G_2^b}$ = (6,3)
- **Observe**: Weight each *entire* sample by the likelihood of the evidence conditioned on the sample
  - Likelihood: $P(\mathbf{E_2^a} \mid \mathbf{G_2^a}) * P(\mathbf{E_2^b} \mid \mathbf{G_2^b})$
- **Resample:** Select prior samples (tuples of values) in proportion to their likelihood