


*Partially observed Markov
Decision Process*



PARTIALLY OBSERVABLE MDP (POMDP)

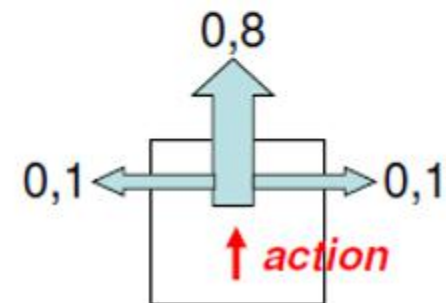
- So far, we have assumed that the environment was fully observable, i.e., the agent always knows which state it is in.
- When the environment is only partially observable, the situation is much less clear
- The agent does not necessarily know which state it is in, so it cannot execute the action $\pi(s)$ recommended for that state.
- Furthermore, the utility of a state s and the optimal action in s depend not just on s , but also on how much the agent knows when it is in s
- For these reasons, partially observable MDPs are usually viewed as much more difficult than ordinary MDPs
- However, we cannot avoid POMDPs because the real world is one

POMDP - 4x3 world (but the agent does perceives its current state)

- **Utility function** for the agent depends on a sequence of states (environment *history*)
- In each state s , the agent receives a **reinforcement** $r(s)$, which may be positive or negative, but must be **bounded**.
- Utility = sum of the rewards received
- Here: $r(s) = -0.04 \quad \forall s$ except $r(4,3) = +1$ and $r(4,2) = -1$

-0,04	-0,04	-0,04	+1 <i>goal</i>
-0,04		-0,04	-1
-0,04	-0,04	-0,04	-0,04
<i>start</i>			

Transition model:



POMDP - 2

- A POMDP has the same elements as an MDP—the transition model $P(s'|s, a)$, actions $A(s)$, and reward function $R(s)$ —but, it also has a sensor model $P(e|s)$
- The sensor model specifies the probability of perceiving evidence e in state s $P(e|s)$
- In POMDPs, the belief state b is a **probability distribution over all possible states**
- For the 4x3 world (version POMDP), the initial belief state could be the uniform distribution over the nine nonterminal states, i.e., $\langle 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 0, 0 \rangle$, $b(s)$ is the probability of being in state s given by the belief state b

POMDP - 3

- The agent can calculate its current belief state as the conditional probability distribution over the actual states given the sequence of percepts and actions so far
- If b was the previous belief state, and the agent does action a and then perceives evidence e , then the new belief state is given by (α is the normalization factor)

$$b'(s') = \alpha P(e|s') \sum_s P(s'|s, a) b(s)$$

- We can define a b' equation like that for each possible state s . Note that we don't need to know the current state, the next belief state b' can be defined as below, given b , a and e
 - $b' = \text{Forward}(b, a, e)$

POMDP - 4

- How can one find the optimal policy in POMDP?
- The fundamental insight required to understand POMDPs is this: the optimal action depends only on the agent's current belief state
- That is, the optimal policy can be described by a mapping $\pi^*(b)$ from belief states to actions.

POMDP agent life cycle

- The decision cycle of a POMDP agent can be broken down into the following three steps:
 1. Given the current belief state b , execute the action $a = \pi^*(b)$.
 2. Receive percept e .
 3. Set the current belief state to

$$b'(s') = \alpha P(e|s') \sum_s P(s'|s, a) b(s)$$

4. go back to step 1
- **Observe** that the optimal policy is defined for **belief states** rather than **states!!!**
 - ...and the agent's action does not define the next belief state!!!

How to find an optimal policy for belief states?

- If we knew the action and the subsequent percept e , we could provide a deterministic update to the belief state, by $b' = \text{Forward}(b, a, e)$
- Of course, the subsequent percept is not known before executing the action, so the agent might arrive in one of several possible belief states b' depending on the percept e that is received.
- We can estimate the probability distribution over b' by..

$$P(b' | b, a) = P(b' | a, b) = \sum_e P(b' | e, a, b) P(e | a, b)$$

- Remember the sum-out

$$P(A) = \sum_i P(A | B_i) P(B_i)$$

and

$$P(A|B) = \sum_i P(A|B, C_i) P(C_i|B)$$

How to estimate $P(e|a,b)=?$

- Let's try to estimate that probability of evidence e given a and b

$$\begin{aligned}P(e|a,b) &= \sum_{s'} P(e|a,s',b)P(s'|a,b) \\ &= \sum_{s'} P(e|s')P(s'|a,b) \\ &= \sum_{s'} P(e|s') \sum_s P(s'|s,a)b(s)\end{aligned}$$

- We use the sum-out and the fact the observations are isolated by the state...

$$\begin{aligned}P(A) &= \sum_i P(A|B_i)P(B_i) \\ &\quad \text{and} \\ P(A|B) &= \sum_i P(A|B,C_i)P(C_i|B)\end{aligned}$$

Estimating transitions between states...

- From the previous expressions for $P(e|a, b)$ and $P(b' | a, b)$

$$\begin{aligned} P(b' | b, a) &= P(b'|a, b) = \sum_e P(b'|e, a, b)P(e|a, b) \\ &= \sum_e P(b'|e, a, b) \sum_{s'} P(e | s') \sum_s P(s' | s, a)b(s) \end{aligned}$$

- Where where $P(b|e, a, b)$ is 1 if $b = \text{Forward}(b, a, e)$ and 0 otherwise
- This equation can be viewed as defining a transition model among belief-states (similar to p for states)

Reward for belief states

- We can also define a reward function for belief states (i.e., the expected reward for the actual states the agent might be in):

$$\rho(b) = \sum_s b(s)R(s)$$

- Together, $P(b|b, a)$ and $\rho(b)$ define an observable MDP on the space of belief states.
- Furthermore, it can be shown that an optimal policy for this MDP, $\pi^*(b)$, is also an **optimal policy for the original POMDP**
- In other words, solving a POMDP on a physical state space can be reduced to solving an MDP on the corresponding belief-state space

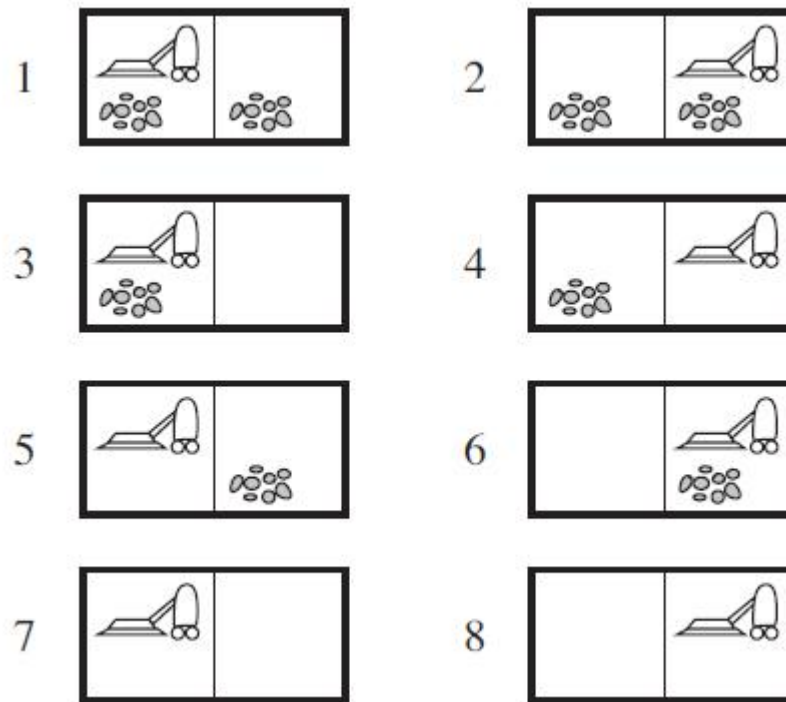
POMDP becomes MDP but...

- Notice that, although we have reduced POMDPs to MDPs, the MDP we obtain has a continuous (and usually **high-dimensional**) state space.
- None of the MDP algorithms described before applies directly to such MDPs
- We will describe a value iteration algorithm designed specifically for POMDPs and an online decision-making algorithm

Value iteration for POMDPs

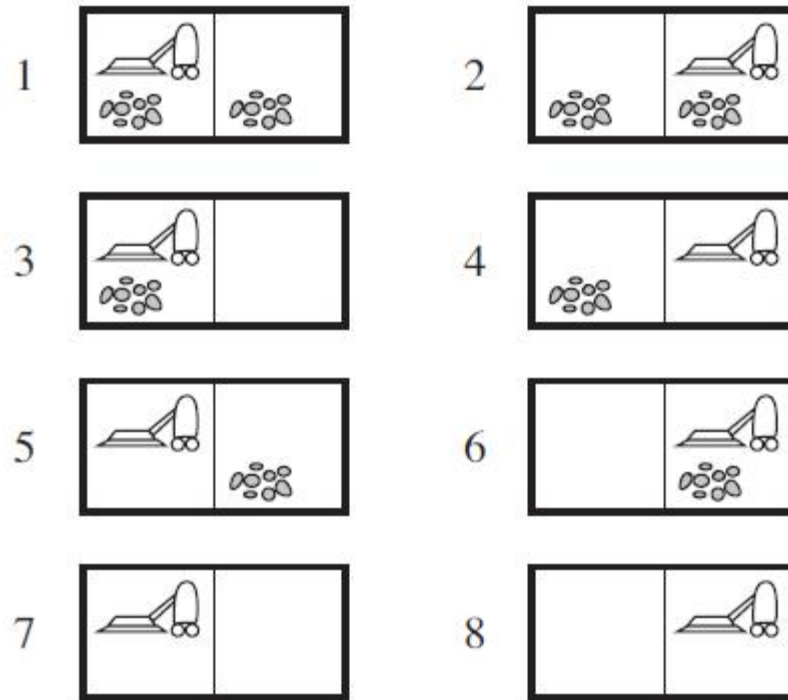
- Consider an optimal policy π^* and its application in a specific belief state b : the policy generates an action, then, for each subsequent percept, the belief state is updated and a new action is generated, and so on
- For one specific b , therefore, the policy is exactly equivalent to a **conditional plan**,
 - Conditional plan is a list of objects, where an object can be an action or a conditional action like If obs= x then Action1 else Action2

Simple Example of POMDP: Vacuum world - the state is not Completely observable



States 7 and 8 are goal states!!

Simple example of POMDP



- Example of Conditional plan (starting on 1, 3, 5 or 7) :

[Suck, Right, if $Bstate = \{6\}$ then Suck else []]

Searching for conditional plans

- As a policy for a given b is a conditional plan, instead of thinking about policies, let us think about **conditional plans** and how the expected utility of executing a fixed conditional plan varies with the initial belief state
- Let the utility of executing a fixed conditional plan p starting in physical state s be $\alpha_p(s)$, Then the expected utility of executing p in belief state b is just

$$\sum_s b(s)\alpha_p(s),$$

- So, the expected utility of a fixed conditional plan varies linearly with b (so it defines a hyperplan in belief space)

Searching for conditional plans -2

- At any given belief state b , the **optimal policy** will choose to execute the conditional plan with highest expected utility;
- and the expected utility of b under the optimal policy is just the utility of that conditional plan:

$$U(b) = U^{\pi^*}(b) = \max_p \sum_s b(s) \alpha_p(s).$$

Searching for conditional plans -3

- If the optimal policy π^* chooses to execute p starting at b , then it is reasonable to expect that it might choose to execute p in belief states that are very close to b ;
- if we bound the **depth** of the conditional plans, then there are only finitely many such plans and the continuous space of belief states will generally be divided into regions, each corresponding to a particular conditional plan that is optimal in that region
- From these two observations, we see that the utility function $U(b)$ on belief states, being the maximum of a collection of hyperplanes, will be **piecewise linear and convex**
 - It allows us to use a recursive approach to search for plans with d -depth, starting with depth one!

Example: Two-state world

- Let's use a simple two-state world. The states are labeled 0 and 1, with $R(0)=0$ and $R(1)=1$.
- There are two actions: **Stay** stays put with probability 0.9 and **Go** switches to the other state with probability 0.9. Let's assume the discount factor $\gamma = 1$
- The sensor reports the correct state with probability 0.6.
- Obviously, the agent should **Stay** when it thinks it's in state 1 and **Go** when it thinks it's in state 0.

Example: Two-state world - 2

- Now let us consider the one-step plans [Stay] and [Go], each of which receives the reward for the current state followed by the (discounted) reward for the state reached after the action

$$\alpha_{[Stay]}(0) = R(0) + \gamma(0.9R(0) + 0.1R(1)) = 0.1$$

$$\alpha_{[Stay]}(1) = R(1) + \gamma(0.9R(1) + 0.1R(0)) = 1.9$$

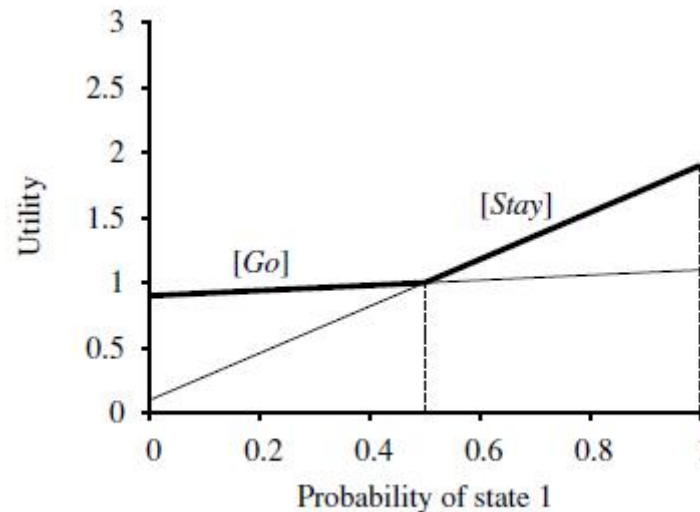
$$\alpha_{[Go]}(0) = R(0) + \gamma(0.9R(1) + 0.1R(0)) = 0.9$$

$$\alpha_{[Go]}(1) = R(1) + \gamma(0.9R(0) + 0.1R(1)) = 1.1$$

- Now, let's check $\sum_s b(s)\alpha_p(s)$ or $b \cdot \alpha_p$

Example: Two-state world - 3

- The hyperplanes (lines, in this case) for $b \cdot \alpha$ [Stay] and $b \cdot \alpha$ [Go] are shown below, their maximum is shown in bold



(a)

- In this case, the optimal one-step policy is to Stay when $b(1) > 0.5$ and Go otherwise

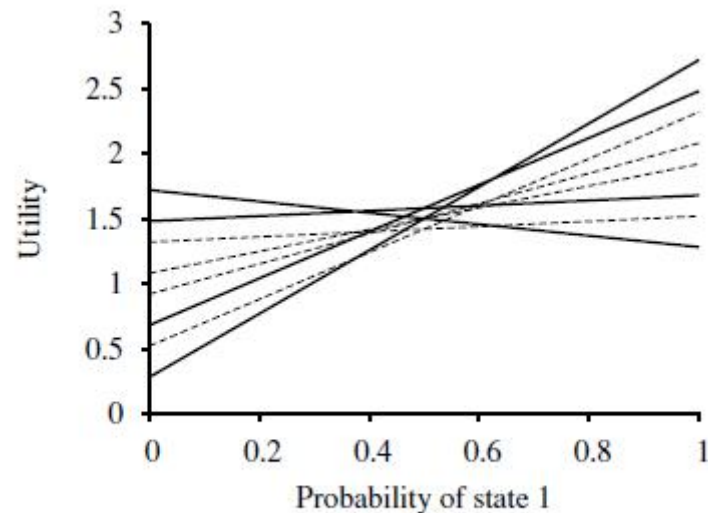
-
- Once we have utilities $u_p(s)$ for all the conditional **plans p of depth 1** in each physical state s ,
 - We can compute the utilities for conditional plans of **depth 2**
 - by considering each possible first action, each possible subsequent percept, and then each way of choosing a depth-1 plan to execute for each percept

[*Stay*; **if** *Percept* = 0 **then** *Stay* **else** *Stay*]

[*Stay*; **if** *Percept* = 0 **then** *Stay* **else** *Go*]

[*Stay*; **if** *Percept* = 1 **then** *Stay* **else** *Stay*] ...

-
- There are eight distinct (Two actions and one percept) depth-2 plans in all, and their utilities are shown below



(b)

-
- So, we can compute depth-1 plans utilities using depth-2 plans utilities and so on...
 - So, We can repeat the process for depth 3, and so on
 - In general, let p be a depth- d conditional plan whose initial action is a
 - and the subsequent depth- $(d - 1)$ subplan for percept e is $p.e$ with utility $\alpha_{p.e}$,
 - It gives us a equation to utility of executing a fixed conditional plan p

$$\alpha_p(s) = R(s) + \gamma \left(\sum_{s'} P(s' | s, a) \sum_e P(e | s') \alpha_{p.e}(s') \right)$$

- That equation (1) gives us a value iteration algorithm

POMDP Value iteration algorithm

function POMDP-VALUE-ITERATION(*pomdp*, ϵ) **returns** a utility function
inputs: *pomdp*, a POMDP with states S , actions $A(s)$, transition model $P(s' | s, a)$,
sensor model $P(e | s)$, rewards $R(s)$, discount γ
 ϵ , the maximum error allowed in the utility of any state
local variables: U , U' , sets of plans p with associated utility vectors α_p

$U' \leftarrow$ a set containing just the empty plan $[\]$, with $\alpha_{[\]}(s) = R(s)$

repeat
 $U \leftarrow U'$
 $U' \leftarrow$ the set of all plans consisting of an action and, for each possible next percept,
 a plan in U with utility vectors computed according to Equation (1)
 $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')

until MAX-DIFFERENCE(U , U') $< \epsilon(1 - \gamma)/\gamma$

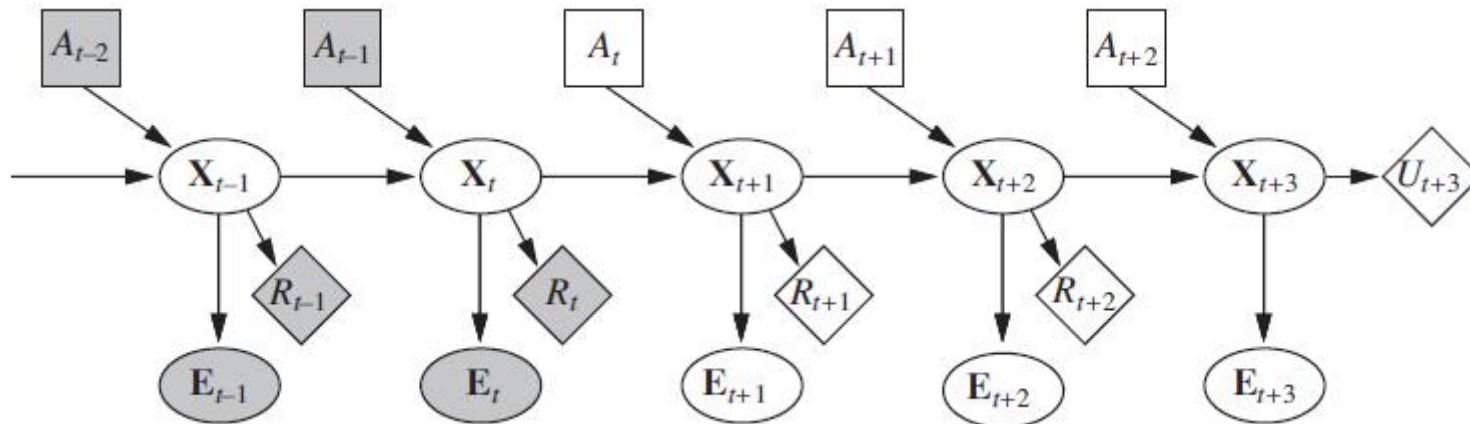
return U

The REMOVE-DOMINATED-PLANS step and MAX-DIFFERENCE test are typically implemented as linear programs

Summary

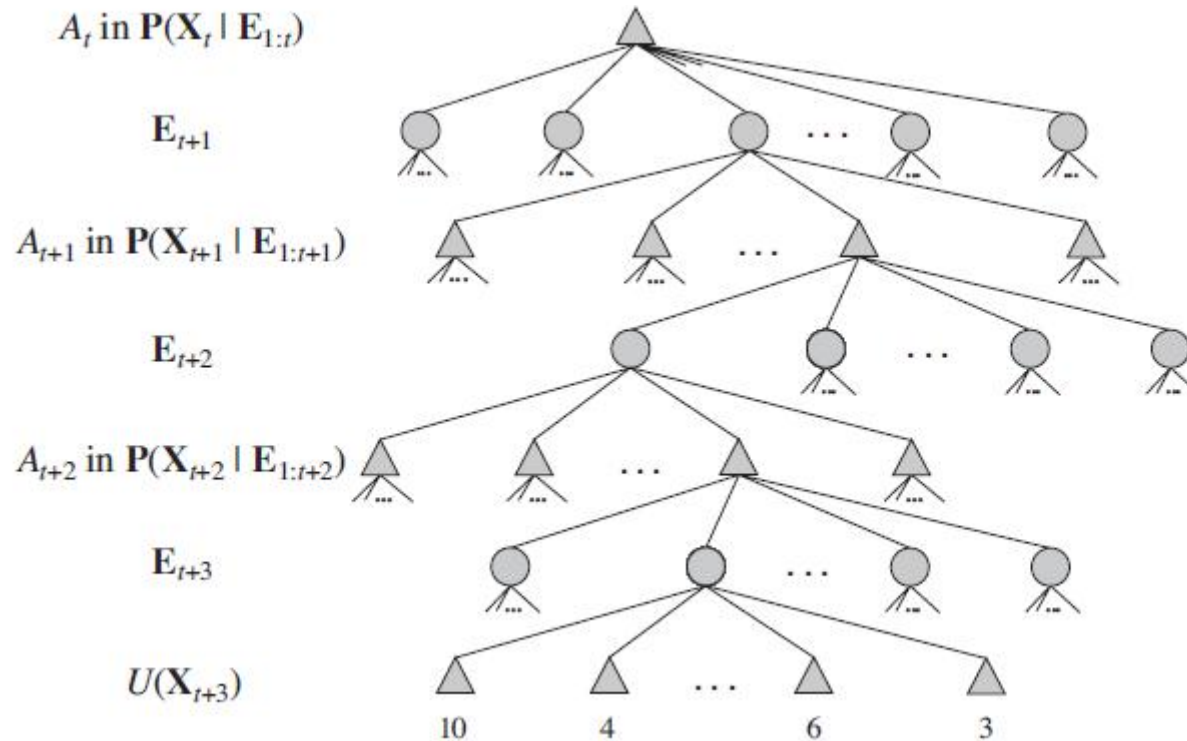
- In practice, the value iteration algorithm is hopelessly inefficient for larger problems
- For general POMDPs, however, finding optimal policies is very difficult (P-SPACE hard, in fact—i.e., very hard indeed)
- Problems with a few dozen states are often infeasible
- We have already studied another approach to deal with these class of problems:
 - Dynamic Bayesian Networks and Dynamic Decision Networks!!

DDN for POMDPs



- The generic structure of a dynamic decision network. Variables with known values are shaded.
- The current time is t and the agent must decide what to do—that is, choose a value for A_t .
- The network has been unrolled into the future for three steps and represents future rewards, as well as the utility of the state at the look-ahead horizon

Decision Tree for DDN



- Part of the look-ahead solution of the DDN shown before.
- Each decision will be taken in the belief state $P(X_t | E_{1:t})$ indicated
- The round (chance) nodes correspond to choices by the environment, namely, what evidence E_{t+i} arrives

-
- A decision can be extracted from the search tree by backing up the utility values from the leaves, taking an average at the chance nodes and taking the maximum at the decision nodes
 - This is similar to the EXPECTIMINIMAX algorithm for game trees with chance nodes, except that
 1. there can also be rewards at non-leaf states and
 2. the decision nodes correspond to belief states rather than actual states
 - Obs. : EXPECTMINIMAX is used for games, here you are playing against one special player **chance**

Summary - 2

- Decision-theoretic agents based on dynamic decision networks have a number of advantages compared with other, simpler POMDP value iteration agent
- In particular, DDN handle partially observable, uncertain environments and can easily revise their “plans” to handle unexpected evidence