Lecture 13: MDP2

Victor R. Lesser
CMPSCI 683
Fall 2010

Today's Lecture

- Continuation with MDP
  - Value and Policy iteration
- Partial Observable MDP (POMDP)

Markov Decision Processes (MDP)

- S - finite set of domain states
- A - finite set of actions
- $P(s' \mid s, a)$ - state transition function
- $R(s)$, $R(s, a)$, or $R(s, a, s')$ - reward function
  - *Could be negative to reflect cost*
- S_0 - initial state
- The Markov assumption:

$$P(s_t \mid s_{t-1}, s_{t-2}, \ldots, s_1, a) = P(s_t \mid s_{t-1}, a)$$

Example: An Optimal Policy

A policy is a choice of what action to choose at each state

An Optimal Policy is a policy where you are always choosing the action that maximizes the "return"/"utility" of the current state

→	→	→	+1
↑		↑	−1
↑	←	←	←

.812	.868	.912	+1
.762		.660	−1
.705	.655	.611	.388

Actions succeed with probability 0.8 and *move at right angles with probability 0.1* (remain in the same position when there is a wall). Actions incur a small cost (0.04).

Computing The Optimal Policy: The Bellman equation

- Optimal policy defined by:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s,a)U(s')$$

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s,a)U(s')$$

- Can be solved using dynamic programming [Bellman, 1957]
  - **How to compute U(j) when it's definition is recursive**

Value iteration [Bellman, 1957]

initialize U'

repeat

  $U \leftarrow U'$

  for each state s do

  $$U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} P(s' \mid s,a)U(s')$$

  end

until *CloseEnough*(U,U')

return greedy policy with respect to U'

Value Iteration Example

Initial Version of U

0.000	0.000	0.000	+1
0.000	0	0.000	-1
0.000	0.000	0.000	0.000

-0.04	-0.0	0.760	+1
-0.04	1	-0.04	-1
-0.04	-0.04	-0.04	-0.04

-0.08	0.560	0.832	+1
-0.08	2	0.464	-1
-0.08	-0.08	-0.08	-0.08

0.392	.738	0.890	+1
-0.12	3	0.572	-1
-0.12	-0.12	0.315	-0.12

0.577	0.819	0.906	+1
0.250	4	0.629	-1
-0.16	0.188	0.394	0.100

0.698	0.849	0.914	+1
0.472	5	0.648	-1
0.162	.313	0.492	0.185

0.809	0.86	0.918	+1
0.754	10	0.660	-1
0.67	0.590	0.577	0.351

0.812	0.868	0.918	+1
0.761	15	0.660	-1
0.704	0.653	0.606	0.378

0.812	0.868	0.918	+1
0.762	19	0.660	-1
0.705	0.655	0.611	0.388

Final Version of U

What is interesting about this example?

What does it say about intermediate reward?

Convergence of VI

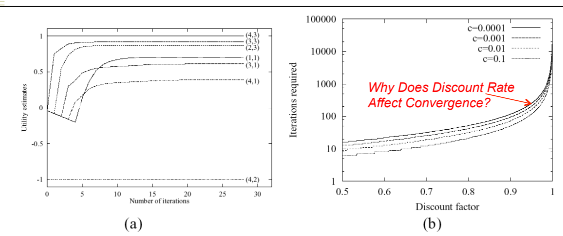


Figure 17.5 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{max}$, for different values of c, as a function of the discount factor γ.

Issues with Value Iteration

- Slow to converge
- Convergence occurs out from goal
- Information about shortcuts propagates out from goal – where there is reward
- Intermediate/Greedy policy is optimal before U values completely settle – Why?.
- Optimal value function is a "fixed point" of VI.

Policy loss

The error bound on the utility of each state may not be the most important factor.

What the agent cares about is how well it does based on a given policy / utility function.

$$\text{if } \| U_i - U^* \| < \varepsilon \text{ then } \| U^{\pi_i} - U^* \| < 2\varepsilon\gamma/(1-\gamma)$$

Note that the policy loss can approach zero long before the utility estimates converge.

Greedy Policy vs Optimal Policy Error bound and policy loss

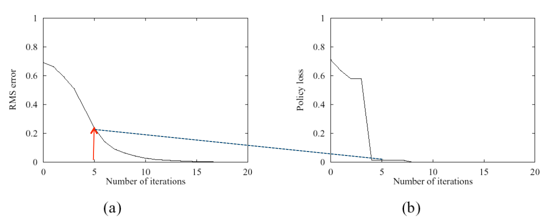


Figure 17.6 (a) The maximum error $\|U_i - U\|$ of the utility estimates after i iterations of value iteration. (b) The policy loss $\|U^{\pi_i} - U\|$ compared to the optimal policy.

Prioritized Sweeping

- State value updates can be performed in any order in value iteration. This suggests trying to decide what states to update to maximize convergence speed.
  - *Use values computed on the current iteration for updates of other values not yet updated on that cycle – how?*
- Prioritized sweeping is a variation of value iteration; more computationally efficient (focused).
- Puts all states in a priority queue in order of how much we think their values might change given a step of value iteration.
- Very efficient in practice (Moore & Atkeson, 1993).

Policy Iteration

- Solve infinite-horizon discounted MDPs in finite time.
  - Start with value function U_0 for each state
  - Let π_1 be greedy policy based on U_0.
  - Evaluate π_1 and *let U_1 be the resulting value function.*
  - Let π_{t+1} be greedy policy for U_t
  - Let U_{t+1} be value of π_{t+1}.
- Each policy is an improvement until optimal policy is reached (another fixed point).
- Since finite set of policies, convergence in finite time.

Policy Iteration

$$\pi_1 \rightarrow V^{\pi_1} \rightarrow \pi_2 \rightarrow V^{\pi_2} \rightarrow \cdots \pi^* \rightarrow V^* \rightarrow \pi^*$$

Policy Evaluation step

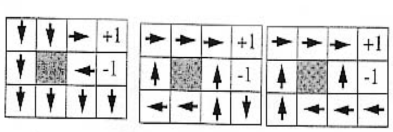
"Greedification" step

Improvement is monotonic

<u>Generalized Policy Iteration</u>:
Intermix the two steps at a finer scale: state by state, action by action, etc.

Simulated PI Example

- Start out with the reward to go (U) of each cell be 0 except for the terminal cells



Fewer iterations than VI, but each iteration more expensive.

Source of disagreement among practioners: PI vs. VI.

Policy iteration [Howard, 1960]

repeat

$\pi \leftarrow \pi'$

$U \leftarrow ValueDetermination(\pi)$ *; reverse from value iteration*

for each state s do

$$\pi'[s] \leftarrow \arg\max_a \sum_{s'} P(s' \mid s, a) U(s')$$

end

until $\pi = \pi'$

Value determination

Can be implemented using:

Value Iteration:

$$U'(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U(s')$$

or

By solving a set of n linear equations:

$$U(s) = R(s) + \sum_{s'} P(s' \mid s, \pi(s)) U(s')$$

Normal Value Iteration

repeat
  $U \leftarrow U'$
  for each state s do
    $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} P(s' \mid s, a) U(s')$
  end
until $CloseEnough(U, U')$

- Notice on each iteration re-computing what the *best action* – convergence to optimal values
- Contrast with the value iteration done in value determination where *policy is kept fixed*. i.e., best action is not changing
  - *convergence to values associated with fixed policy much faster*

Adding in Time to MDP Actions SMDP

- <S,A,P,R>
  - S: states
  - A: actions
  - P(s,a,N,s'): likelihood of transition at a certain time step N
  - R(s): reward

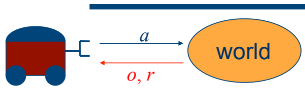
$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) U(s')$$

N

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) U(s')$$

P (s'|s,N,a)

POMDP (Partially Observable MDP)



- The agent does not fully observe the state
- Current state is not enough to make the optimal decision anymore
- ***Need entire observation sequence to guarantee the Markovian property***

The POMDP Model

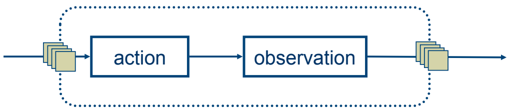
Augmenting the completely observable MDP with the following elements:

- O – a finite set of observations
- P(o|s',a) – observation function: the probability that o is observed after taking action a resulting in a transition to state s'
- A discrete probability distribution over starting states (the initial belief state):

$$b_0 = \{b_0(0), b_0(1), \ldots, b_0(|S|-1)\}$$

POMDP Policy

- Policy:



Observation sequence so far: a distribution of belief in the current world state

Stage t

The new observation sequence

Performance criteria and utility function

- We will focus on infinite-horizon problems
  → performance criterion = expected discounted reward over an infinite horizon

$$E_{b_0}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right]$$

A specific policy generates a set of possible histories, each with its own likelihood and reward

Utility function measurement:

- b_0 is the *a priori* state probability distribution
- γ is the discount factor

Policy representation

- A policy π is a rule for selecting actions
- For MDPs this can simply be a mapping from states (of the underlying system) to actions
- For POMDPs this is not possible, because the system state is only partially observable
- Thus, a policy must map from a "decision state" to actions. This "decision state" can be defined by:
  - The history of the process (action, observation sequence)
    - (Problem: grows exponentially, not suitable for infinite horizon problems)
  - A probability distribution over states
  - The memory of a finite-state controller

Bayesian policies (1)

- The whole history of the process is saved in a probability distribution over all system states
- This probability vector called *belief state* can be updated by Bayesian conditioning after each action and observation --
- b(s) denotes the probability that the current state of the system is s
  - b is the vector of probs over all s, called the belief state
- P (s',o|s,a) = P(s'|s,a)P(o|s',a)
- b^a_o(s') = Sum_sP(s',o|s,a)b(s)/ $Sum_{s,s'}$P(s',o|s,a)b(s)

Bayesian policies (2)

- A belief state updated by Bayesian conditioning is a sufficient statistic that summarizes all relevant information about the history.
- We can define an MDP with a state set consisting of all possible belief states thus mapping a POMDP into an MDP
- $V'(b_i)=\max_a\{r(b_i,a)+\gamma*(\text{sum}_o P(o|b_i,a)V(b_i{}^a{}_o)\}$ where
  $r(b_i,a) = \text{sum}_s b_i(s)r(s,a)$
- The set of belief states is **continuous** and **infinite** but this problem can be *fixed by using a set of real number basis vectors of size |S| to represent V since DP preserves the piecewise linearity and convexity of the value function.*

Finite-memory policies (1)

- We want a discrete representation with a finite number of states!
  - Could do simple binning of probabilities of states but this may be a very poor approximation
  - Does not reflect which differences are important and those that are not relevant
- A finite state controller maps H*, the set of all possible histories, into a finite number of memory states.
- Unlike a belief state, a memory state is not a sufficient statistic but as the number of memory states is finite, the policy representation becomes easier.

Finite-state controllers

- Finite set of inputs – the set of possible observations O after each action
- Finite set of outputs – the set of actions A
- A finite set of memory states Q
- A memory state update function $\tau : Q \times O \rightarrow Q$
- An output function (the policy) $\alpha : Q \rightarrow A$
- A nonempty set of possible starting memory states + a rule for selecting the starting one
- A possibly empty set of final memory states

Difficulties of the finite-memory approach

- The memory state is not necessarily a sufficient statistic, but:
  - A finite-state controller can perform arbitrarily close to optimal by using arbitrarily many memory states
  - Mapping different histories into the same memory state is a form of generalization in which marginally relevant information is ignored → focus on the most relevant aspects of the history
- How to find a good finite-memory representation?
  - Finding the best finite-memory representation is the difficulty of determining how to organize limited memory and use it effectively in decision making, i.e. deciding what to remember and what to forget.

Policy evaluation for (PO)MDPs

- Utility function: $U^{\pi}(b_0) = E\left[\sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t))\right]$
- For completely observable MDPs a *policy determines a Markov chain*
  - *each state corresponds to a state of the MDP with associated action and transition probabilities to next states.*
- Then the utility of each state can be determined by solving a system of |S| linear equations:

$$U^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} \Pr(s' \mid s, \pi(s)) U^{\pi}(s'), \forall s \in S$$

Policy evaluation for POMDPs (2)

- We allow the finite-state controller to visit an infinite number of belief states.
- In this way, the finite-state controller determines a Markov chain in which each state corresponds to a combination of a memory state q_i and a system state s_j.
  - q_i represents an approximation of the history of observations and actions that were taken to get to state s_j
- Thus, the size of the Markov chain is |Q||S|.

Policy evaluation for POMDPs (3)

two state POMDP becomes a four state markov chain.

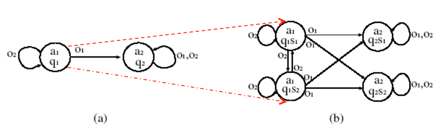


Figure 2.4. (a) Finite-state controller and (b) corresponding Markov chain for two-state, two-action and two observation POMDP.

By Mapping a finite controller into a Markov Chain can be used to compute utility of finite controller of POMDP; can then have a search process to find finite controller that maximizes utility of POMDP

Next Lecture

- Decision Making As An Optimization Problem