Artificial Intelligence **Spring 2005**
Professor Greenwald **Lecture #11**

Markov Decision Processes: Control[1]

In this lecture, we extend our discussion of Markov reward processes to Markov decision processes (MDP). For MDPs, we pose and solve the control problem. Specifically, we describe value iteration and policy iteration, two dynamic programming algorithms that are used to compute an optimal policy in an MDP.

1 Definition and An Example

Recall the following:

A *stochastic process* is a sequence of random variables $\{X_t\}_{t=0}^{\infty}$. A stochastic process $\{X_t\}_{t=0}^{\infty}$ induces a probability transition function of the form $P[X_{t+1} = s_{t+1}|X_t = s_t, \ldots, X_0 = s_0]$: i.e., the probability that the state at future time $t+1$ is s_{t+1}, given that the states at past times $t, \ldots, 0$ were $s_t, \ldots, s_0$, respectively.

A *Markov process* is a stochastic process *s.t.* for all t, for all $s_0, \ldots, s_t, s_{t+1}$,

$$P[X_{t+1} = s_{t+1}|X_t = s_t, \ldots, X_0 = s_0] = P[X_{t+1} = s_{t+1}|X_t = s_t] \qquad (1)$$

Equation 1 is the *Markov property*, sometimes called the memoryless property; it implies that probability transitions to future states, such as s_{t+1}, depend only on the present state s_t, but are independent of the remote past, $s_{t-1}, \ldots, s_0$.

1.1 Markov Decision Processes

An agent operating in a non-deterministic environment transitions from state to state, in general making decisions and obtaining rewards along the way, as follows: at time t,

1. state is s_t
2. choose action a_t
3. receive reward r_t
4. transition to state s_{t+1} with probability $P[s_{t+1}|s_t, a_t]$

Markov decision processes (MDPs) model such agent-environment interactions. A (discrete-time) *Markov decision process* is a tuple $\langle S, A, R, P\rangle$, where time is discrete: i.e., $t \in T = \{0, 1, \ldots\}$, and

- S is a finite set of states ($s \in S$)
- A is a finite set of actions ($a \in A$)
- $R : S \times A \to \mathbb{R}$ is a reward function
- $P : S \times A \to \Delta(S)$ is a probability transition function (or matrix)
  $\Delta(S)$ is the set of probability distributions over S

Example The TAC Classic flight auctions (in isolation) are examples of MDPs. Let us simplify one TAC Classic flight auction and model it as an MDP.

The state is defined in terms of the price of the flight and the time remaining until the end of the auction. Specifically, the state space is the cross product of the set of possible prices, say $\mathcal{P} = \{150, 160, \ldots, 590, 600\}$ and the time, which we assume varies discretely from $t = 0$ through time $T = 30$, unioned with a designated state END. Let p_t denote the price at time t.

The set of possible actions A includes *buy now* (B) and *(re)consider later* (C). Rewards depend on the flight's valuation. Assuming v represents this valuation, $R(p_t, B) = v - p_t$ and $R(p_t, C) = 0$, for all p_t; in addition, $R(\text{END}, a) = 0$, for all actions $a \in A$. Finally, transition probabilities depend on states and actions: for all prices $p \in \mathcal{P}$, actions $a \in A$, and times $t \in \{0, \ldots, T\}$,

$$\begin{aligned} P[\text{END}|p_t = p, a_t = B] &= 1.0 \\ P[p_{t+1} = p + 10|p_t = p, a_t = C] &= 0.5 \\ P[p_{t+1} = p - 10|p_t = p, a_t = C] &= 0.5 \\ P[\text{END}|\text{END}, a_t = a] &= 1.0 \\ P[\text{END}|p_T = p, a_T = a] &= 1.0 \end{aligned}$$

At state p_t, what is the optimal action?

2 State Values

A *policy*[2] is a map from states to actions: i.e., $\pi : S \to A$. By Bellman's theorem, the state value $V^\pi(s)$ at state s under policy π is the sum of the immediate reward obtained in state s and the discounted sum of the rewards obtained by following policy π thereafter:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{s'}[V^\pi(s')] \tag{2}$$

Policy π dominates policy $\hat{\pi}$ (notation $\pi \gg \hat{\pi}$) iff $V^\pi(s) \geq V^{\hat{\pi}}(s)$ for all states $s \in S$. We seek an optimal policy: i.e., π^* *s.t.* $\pi^* \gg \pi$, for all policies π.

[2]It suffices to restrict our attention to *deterministic, stationary* policies π, in which the same pure (i.e., non-randomized) action is taken every time state s is visited.

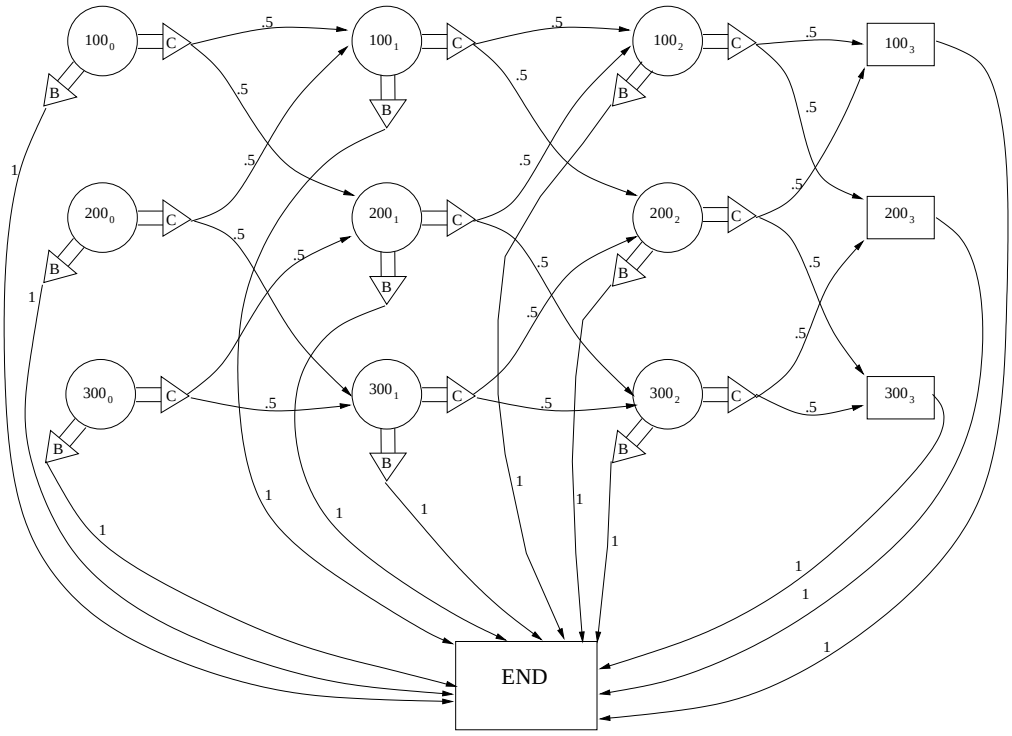

Figure 1: TAC Classic Flight Auctions as an MDP: States are indicated by circles. Fat arrows indicate actions; they are labeled with rewards. Skinny arrows indicate transitions; they are labeled with probabilities.

That such a policy exists is not at all obvious. Indeed, Bellman's celebrated *optimality* equations (Equation 3) and their accompanying dynamic program establish both the existence of an optimal value function and a corresponding optimal policy, as well as an efficient technique for computing them:

$$V(s) = \max_a \{R(s,a) + \gamma \mathbb{E}_{s'}[V(s')]\} \tag{3}$$

As in the case of MRPs, to find a solution to this system of ($|S|$) equations (with $|S|$ unknowns), we rely on Banach's fixed point theorem. The optimal value function V^* is the unique solution to this system of equations. The optimal policy π^* maps state s into an optimal action, as follows:

$$\pi^*(s) \in \arg\max_a \{R(s,a) + \gamma \mathbb{E}_{s'}[V^*(s')]\} \tag{4}$$

Exercise Show that the mapping implicit in Equation 3 is a contraction on $(\mathbb{R}^S, L_\infty)$.

3 Action Values

The action value $Q^\pi(s, a)$ associated with state s and action a is defined as the sum of the immediate reward obtained by taking action a in state s and the discounted sum of the rewards obtained by following policy π thereafter:

$$Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s'}[V^\pi(s')] \tag{5}$$

Restating Bellman's optimality equations in terms of action values yields:

$$Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s'}[V(s')] \tag{6}$$

$$V(s) = \max_a Q(s, a) \tag{7}$$

Simplifying,

$$Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s'}[\max_a Q(s', a)] \tag{8}$$

As above, to find a solution to this system of ($|S \times A|$) equations (with $|S \times A|$ unknowns), we rely on Banach's fixed point theorem. The optimal action-value function Q^* is the unique solution to this system of equations.

Exercise Show that the mapping implicit in Equation 8 is a contraction on $(\mathbb{R}^S, L_\infty)$.

Given Q^*, the optimal policy π^* maps state s into an optimal action, as follows:

$$\pi^*(s) \in \arg\max_a Q^*(s, a) \tag{9}$$

While the optimal action-value function Q^* is unique, the optimal policy π^* need not be unique.

4 Value Iteration

The value iteration algorithm, which is based on Equation 3, updates as follows:

$$V(s) \leftarrow \max_a \{R(s, a) + \gamma \sum_{s'} P[s'|s, a]V(s')\} \tag{10}$$

Equivalently, value iteration can be described in terms of Equations 6 and 7:

$$Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} P[s'|s, a]V(s') \tag{11}$$

$$V(s) \leftarrow \max_a Q(s, a) \tag{12}$$

The algorithm, which is depicted in Table 1, first computes the value of each state for all actions, and then sets each state's value to be the greatest value achieved among all courses of action. The actions that yield the optimal state values can be extracted as the optimal policy.

VALUE_ITERATION(MDP, γ, ϵ)

Inputs	discount factor γ
	convergence test ϵ
Output	optimal state-value function V^*
Initialize	$V = 0$ and $V' = \infty$

`while` $\max_s |V(s) - V'(s)| > \epsilon$ `do`

1. $V' = V$
2. for all $s \in S$
   (a) for all $a \in A$
      i. $Q(s,a) = R(s,a) + \gamma \sum_{s'} P[s'|s,a]V(s')$
   (b) $V(s) = \max_a Q(s,a)$

`return` V

Table 1: Value Iteration *á la* Gauss-Seidel.

4.1 Example: TAC Flight Auctions

The following tables depict the computation of state and action values and the optimal policy in a TAC flight auction, assuming 3 prices, namely \$100, \$200, and \$300, and 4 time steps, with $V = 500$ and $\gamma = 1$.

$Q(s,a)$	$t=0$		$t=1$		$t=2$		$t=3$	
	B	C	B	C	B	C	B	C
300	200	300	200	275	200	250	200	0
200	300	337.5	300	325	300	300	300	0
100	400	362.5	400	350	400	350	400	0

$V(s)$	$t=0$	$t=1$	$t=2$	$t=3$
300	300	275	250	200
200	337.5	325	300	300
100	400	400	400	400

$\pi(s)$	$t=0$	$t=1$	$t=2$	$t=3$
300	C	C	C	B
200	C	C	C/B	B
100	B	B	B	B

The optimal policy prescribes that an agent buy whenever the price hits the lower bound. Regardles of price, an agent should buy if all time has elapsed. Otherwise, if time remains and the price is not rockbottom, it is optimal to consider buying later, since there is some chance of seeing the price drop.

5 Policy Iteration

Policy iteration is a two-phase dynamic programming method for computing optimal policies directly. The first phase, *policy evaluation*, computes the state values for the current (fixed) policy via Equation 2. The second phase, *policy improvement*, improves upon the current policy (whenever possible) in a greedy fashion. Policy improvement updates based on Equations 5 and 9.

In practice, value iteration is faster than policy iteration *per iteration*; however, policy iteration takes far fewer iterations to converge. One modified version of policy iteration does not wait for the policy evaluation phase of policy iteration to converge, and instead produces approximations of V^π. This modification leads to substantial speedups in the runtime of policy iteration.

5.1 Policy Evaluation

Policy evaluation in Markov decision processes computes state values given some policy exactly as state values are evaluated in Markov processes:

$$V^\pi(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} P[s'|s, \pi(s)] V^\pi(s') \tag{13}$$

5.2 Policy Improvement

The soundness of policy iteration follows from the *policy improvement theorem*:

Theorem Given policies π_1 and π_2, if $Q^{\pi_1}(s, \pi_2(s)) \geq Q^{\pi_1}(s, \pi_1(s)) = V^{\pi_1}(s)$ for all states $s \in S$, then $V^{\pi_2}(s) = Q^{\pi_2}(s, \pi_2(s)) \geq V^{\pi_1}(s)$ for all $s \in S$.

Proof (Sketch)

$$\begin{aligned}
V^{\pi_1}(s) &= Q^{\pi_1}(s, \pi_1(s)) \\
&\leq Q^{\pi_1}(s, \pi_2(s)) \\
&= R(s, \pi_2(s)) + \gamma \mathbb{E}_{s'}[V^{\pi_1}(s')] \\
&= R(s, \pi_2(s)) + \gamma \mathbb{E}_{s'}[Q^{\pi_1}(s', \pi_1(s'))] \\
&\leq R(s, \pi_2(s)) + \gamma \mathbb{E}_{s'}[Q^{\pi_1}(s', \pi_2(s'))] \\
&= R(s, \pi_2(s)) + \gamma \mathbb{E}_{s'}[R(s', \pi_2(s')) + \gamma \mathbb{E}_{s''}[V^{\pi_1}(s'')]] \\
&= R(s, \pi_2(s)) + \gamma \mathbb{E}_{s'}[R(s', \pi_2(s'))] + \gamma^2 [\mathbb{E}_{s''}[V^{\pi_1}(s'')]] \; = \; \cdots \; = \; V^{\pi_2}(s)
\end{aligned}$$

The policy improvement steps in the policy iteration algorithm are as follows:

$$Q^\pi(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} P[s'|s, a] V^\pi(s') \tag{14}$$

$$\pi(s) \in \arg\max_a Q(s, a) \tag{15}$$

POLICY_ITERATION(MDP, γ, ϵ)
Inputs discount factor γ
convergence test ϵ
Output optimal policy π^*
Initialize $\pi \neq \pi'$

`while` $\pi \neq \pi'$ `do`

1. $\pi' = \pi$
2. $V^\pi =$ POLICY_EVALUATION(MDP, π, γ, ϵ)
3. $\pi =$ POLICY_IMPROVEMENT(MDP, V^π, γ)

`return` π

POLICY_EVALUATION(MDP, π, γ, ϵ)
Inputs policy π
discount factor γ
convergence test ϵ
Output state-value function V^π
Initialize $V = 0$ and $V' = \infty$

`while` $\max_s |V(s) - V'(s)| > \epsilon$ `do`

1. $V' = V$
2. for all $s \in S$
   (a) $V(s) = R(s, \pi(s)) + \gamma \sum_{s'} P[s'|s, \pi(s)]V(s')$

`return` V

POLICY_IMPROVEMENT(MDP, V, γ)
Inputs value function V
discount factor γ
Output improved policy π

for all $s \in S$

1. for all $a \in A$
   (a) $Q(s, a) = R(s, a) + \gamma \sum_{s'} P[s'|s, a]V(s')$
2. $\pi(s) \in \arg\max_a Q(s, a)$

`return` π

Table 2: Policy Iteration.

5.3 Example: TAC Flight Auctions

The following tables depict the iterative computation of policies, state, and action values in one TAC flight auction. The flight's valuation is 500.

Initialization

π	$t=0$	$t=1$	$t=2$	$t=3$
300	B	B	B	B
200	B	B	B	B
100	B	B	B	B

Iteration 0

V^π	$t=0$	$t=1$	$t=2$	$t=3$
300	200	200	200	200
200	300	300	300	300
100	400	400	400	400

$Q^\pi(s,a)$	$t=0$		$t=1$		$t=2$		$t=3$	
	B	C	B	C	B	C	B	C
300	200	250	200	250	200	250	200	0
200	300	300	300	300	300	300	300	0
100	400	350	400	350	400	350	400	0

π	$t=0$	$t=1$	$t=2$	$t=3$
300	C	C	C	B
200	C/B	C/B	C/B	B
100	B	B	B	B

Iteration 1

V^π	$t=0$	$t=1$	$t=2$	$t=3$
300	287.5	275	250	200
200	300	300	300	300
100	400	400	400	400

$Q^\pi(s,a)$	$t=0$		$t=1$		$t=2$		$t=3$	
	B	C	B	C	B	C	B	C
300	200	287.5	200	275	200	250	200	0
200	300	337.5	300	325	300	300	300	0
100	400	350	400	350	400	350	400	0

π	$t=0$	$t=1$	$t=2$	$t=3$
300	C	C	C	B
200	C	C	C/B	B
100	B	B	B	B

Iteration 2

V^π	$t=0$	$t=1$	$t=2$	$t=3$
300	300	275	250	200
200	337.5	325	300	300
100	400	400	400	400

$Q^\pi(s,a)$	$t=0$		$t=1$		$t=2$		$t=3$	
	B	C	B	C	B	C	B	C
300	200	300	200	275	200	250	200	0
200	300	337.5	300	325	300	300	300	0
100	400	362.5	400	350	400	350	400	0

π	$t=0$	$t=1$	$t=2$	$t=3$
300	C	C	C	B
200	C	C	C/B	B
100	B	B	B	B

As the new policy does not differ from the old, policy iteration has converged. Note that the current values of V^π represent the values of the optimal policy.

Problems

#1 Consider the following controlled version of *Gambler's Ruin* in which the gambler places bets on the outcome of a biased coin flip. Assume the gambler's worth is between 0 and N (i.e., $S = \{0, 1, \ldots, N\} \cup \{\text{END}\}$). At each state s, the gambler stakes some amount n in the range $A = \{1, \ldots, \min\{s, N-s\}\}$. The coin turns up heads with probability p and tails with probability $1-p$. If the coin turns up heads, the gambler wins the amount he stakes (i.e., he transitions to state $s+n$); otherwise, the gambler loses the amount he stakes (i.e., he transitions to state $s-n$). A reward of 1 is received upon transitioning to state N; all other rewards are 0. The gambler transitions from states 0 and N to the absorbing state END, deterministically. The constraints on the gambler's range of actions ensure that (i) he stakes at least \$1 but no more than his worth; (ii) he stakes no more than $N-s$, preventing his worth from ever exceeding N.

(a) Draw the corresponding Markov decision process, assuming the gambler's worth is limited to \$5 (i.e., $N = 5$).

(b) What is the optimal policy if $p < 0.5$, $N = 100$, and $\gamma = 1$. Why? (Solve this problem by implementing value iteration or policy iteration.)

(c) What is the optimal policy if $p > 0.5$, $N = 100$, and $\gamma = 1$. Why?

#2 The goal of this question is to extend the MINIMAX algorithm to games of chance, like backgammon and monopoly.

(a) Formally define games of chance by extending the definition of game trees.

(b) Extend the MINIMAX algorithm to take as input a game of chance.

#3 The goal of this question is to extend value and policy iteration to games of chance, like backgammon and monopoly.

(a) Formally define games of chance by extending the definition of MDPs.

(b) Extend the value iteration algorithm to take as input a game of chance.

(c) Extend the policy iteration algorithm to take as input a game of chance.

#4 "Pig" is a two-player children's dice game. Each player starts with a total score of zero, which is increased on each turn by dice rolling. The first to reach 50 (or more) wins. On his turn, a player accumulates a subtotal by repeatedly rolling a 6-sided die. If ever he rolls a 1, however, he loses the subtotal and only 1 is added to his running total. Thus, before each roll, each player decides between two actions: add his current subtotal to his running total and pass the turn to the other player; or continue rolling, risking an unlucky 1.

(a) Formulate "Pig" as a game of chance.

(b) Suppose Bob plays the following fixed strategy: roll exactly once, add the value on the die to his total, and pass the turn back to his opponent, Alice. Solve for Alice's optimal policy against Bob. What is her probability of winning assuming she goes first and the game is played until 10, 20, or 50?

(c) Now suppose Bob wises up and plays the minimax strategy. If Alice also plays optimally (i.e., minimax), what is her probability of winning assuming she goes first and the game is played until 10, 50, or 100?