CS 188
Fall 2019

Section Handout 4 Solutions

The preamble is an abbreviation of the lecture notes

Markov Decision Processes

A Markov Decision Process is defined by several properties:

- A set of states S
- A set of actions A.
- A start state.
- Possibly one or more terminal states.
- Possibly a **discount factor** γ.
- A **transition function** $T(s, a, s')$.
- A **reward function** $R(s, a, s')$.

The Bellman Equation

- $V^*(s)$ – the optimal value of s is the expected value of the utility an optimally-behaving agent that starts in s will receive, over the rest of the agent's lifetime.
- $Q^*(s, a)$ - the optimal value of (s, a) is the expected value of the utility an agent receives after starting in s, taking a, and acting optimally henceforth.

Using these two new quantities and the other MDP quantities discussed earlier, the Bellman equation is defined as follows:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

We can also define he equation for the optimal value of a q-state (more commonly known as an optimal **q-value**):

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

which allows us to reexpress the Bellman equation as

$$V^*(s) = \max_a Q^*(s, a).$$

Value Iteration

The **time-limited value** for a state s with a time-limit of k timesteps is denoted $V_k(s)$, and represents the maximum expected utility attainable from s given that the Markov decision process under consideration terminates in k timesteps. Equivalently, this is what a depth-k expectimax run on the search tree for a MDP returns.

Value iteration is a **dynamic programming algorithm** that uses an iteratively longer time limit to compute time-limited values until convergence (that is, until the V values are the same for each state as they were in the past iteration: $\forall s, V_{k+1}(s) = V_k(s)$). It operates as follows:

1. $\forall s \in S$, initialize $V_0(s) = 0$. This should be intuitive, since setting a time limit of 0 timesteps means no actions can be taken before termination, and so no rewards can be acquired.
2. Repeat the following update rule until convergence:

$$\forall s \in S,\ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

   At iteration k of value iteration, we use the time-limited values for with limit k for each state to generate the time-limited values with limit $(k + 1)$. In essence, we use computed solutions to subproblems (all the $V_k(s)$) to iteratively build up solutions to larger subproblems (all the $V_{k+1}(s)$); this is what makes value iteration a dynamic programming algorithm.

1 MDPs: Micro-Blackjack

In micro-blackjack, you repeatedly draw a card (with replacement) that is equally likely to be a 2, 3, or 4. You can either Draw or Stop if the total score of the cards you have drawn is less than 6. If your total score is 6 or higher, the game ends, and you receive a utility of 0. When you Stop, your utility is equal to your total score (up to 5), and the game ends. When you Draw, you receive no utility. There is no discount ($\gamma = 1$). Let's formulate this problem as an MDP with the following states: $0, 2, 3, 4, 5$ and a *Done* state, for when the game ends.

(a) What is the transition function and the reward function for this MDP? The transition function is

$$
\begin{aligned}
T(s, Stop, Done) &= 1 \\
T(0, Draw, s') &= 1/3 \text{ for } s' \in \{2, 3, 4\} \\
T(2, Draw, s') &= 1/3 \text{ for } s' \in \{4, 5, Done\} \\
T(3, Draw, s') &= \begin{cases} 1/3 \text{ if } s' = 5 \\ 2/3 \text{ if } s' = Done \end{cases} \\
T(4, Draw, Done) &= 1 \\
T(5, Draw, Done) &= 1 \\
T(s, a, s') &= 0 \text{ otherwise}
\end{aligned}
$$

The reward function is

$$
\begin{aligned}
R(s, Stop, Done) &= s, s \leq 5 \\
R(s, a, s') &= 0 \text{ otherwise}
\end{aligned}
$$

(b) Fill in the following table of value iteration values for the first 4 iterations.

States	0	2	3	4	5
V_0	0	0	0	0	0
V_1	0	2	3	4	5
V_2	3	3	3	4	5
V_3	10/3	3	3	4	5
V_4	10/3	3	3	4	5

(c) You should have noticed that value iteration converged above. What is the optimal policy for the MDP?

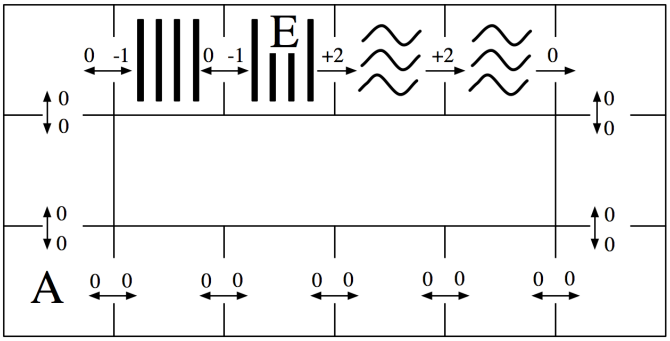
States	0	2	3	4	5
π^*	Draw	Draw	Stop	Stop	Stop

(d) Perform one iteration of policy iteration for one step of this MDP, starting from the fixed policy below:

States	0	2	3	4	5
π_i	Draw	Stop	Draw	Stop	Draw
V^{π_i}	2	2	0	4	0
π_{i+1}	Draw	Stop	Stop	Stop	Stop

Q2. MDPs: Grid-World Water Park

Consider the MDP drawn below. The state space consists of all squares in a grid-world water park. There is a single waterslide that is composed of two ladder squares and two slide squares (marked with vertical bars and squiggly lines respectively). An agent in this water park can move from any square to any neighboring square, unless the current square is a slide in which case it must move forward one square along the slide. The actions are denoted by arrows between squares on the map and all deterministically move the agent in the given direction. The agent cannot stand still: it must move on each time step. Rewards are also shown below: the agent feels great pleasure as it slides down the water slide (+2), a certain amount of discomfort as it climbs the rungs of the ladder (-1), and receives rewards of 0 otherwise. The time horizon is infinite; this MDP goes on forever.



(a) How many (deterministic) policies π are possible for this MDP?

2^{11}

(b) Fill in the blank cells of this table with values that are correct for the corresponding function, discount, and state. *Hint: You should not need to do substantial calculation here.*

	γ	$s = A$	$s = E$
$V_3^*(s)$	1.0	0	4
$V_{10}^*(s)$	1.0	2	4
$V_{10}^*(s)$	0.1	0	2.2
$Q_1^*(s, \text{west})$	1.0	———	0
$Q_{10}^*(s, \text{west})$	1.0	———	3
$V^*(s)$	1.0	∞	∞
$V^*(s)$	0.1	0	2.2

$V_{10}^*(A), \gamma = 1$: In 10 time steps with no discounting, the rewards don't decay, so the optimal strategy is to climb the two stairs (-1 reward each), and then slide down the two slide squares (+2 rewards each). You only have time to do this once. Summing this up, we get $-1 - 1 + 2 + 2 = 2$.

$V_{10}^*(E), \gamma = 1$: No discounting, so optimal strategy is sliding down the slide. That's all you have time for. Sum of rewards $= 2 + 2 = 4$.

$V_{10}^*(A), \gamma = 0.1$. The discount rate is 0.1, meaning that rewards 1 step further into the future are discounted by a factor of 0.1. Let's assume from A, we went for the slide. Then, we would have to take the actions $A \to B, B \to C, C \to D, D \to E, E \to F, F \to G$. We get the first -1 reward from $C \to D$, discounted by γ^2 since it is two actions in the future. $D \to E$ is discounted by γ^3, $E \to F$ by γ^4, and $F \to G$ by γ^5. Since γ is low, the positive rewards you get from the slide have less of an effect as the larger negative rewards you get from climbing up. Hence, the sum of rewards of taking the slide path would be negative; the optimal value is 0.

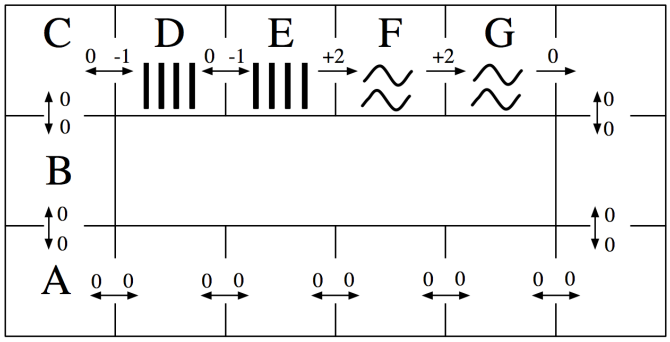
$V_{10}^*(E), \gamma = 0.1$. Now, you don't have to do the work of climbing up the stairs, and you just take the slide down. Sum of rewards would be 2 (for $E \to F$) + 0.2 (for $F \to G$, discounted by 0.1) = 2.2.

$Q_{10}^*(E, west), \gamma = 1$. Remember that a Q-state (s,a) is when you start from state s and are committed to taking a. Hence, from E, you take the action West and land in D, using up one time step and getting an immediate reward of 0. From D, the optimal strategy is to climb back up the higher flight of stairs and then slide down the slide. Hence, the rewards would be $-1(D \to E) + 2(E \to F) + 2(F \to G) = 3$.

$V^*(s), \gamma = 1$. Infinite game with no discount? Have fun sliding down the slide to your content from anywhere.

$V^*(s), \gamma = 0.1$. Same reasoning apply to both A and E from $V_{10}^*(s)$. With discounting, the stairs are more costly to climb than the reward you get from sliding down the water slide. Hence, at A, you wouldn't want to head to the slide. From E, since you are already at the top of the slide, you should just slide down.

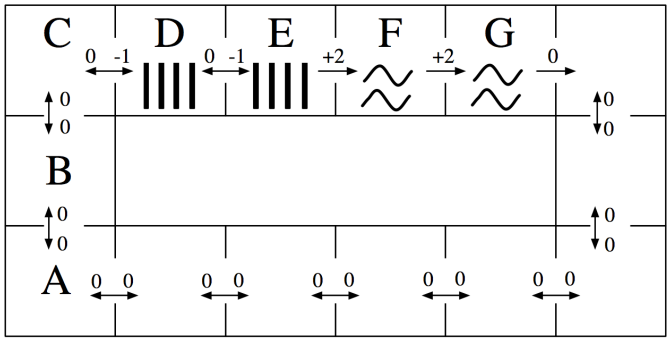
Use this labeling of the state space to complete the remaining subproblems:



(c) Fill in the blank cells of this table with the Q-values that result from applying the Q-update for the transition specified on each row. You may leave Q-values that are unaffected by the current update blank. Use discount $\gamma = 1.0$ and learning rate $\alpha = 0.5$. Assume all Q-values are initialized to 0. (Note: the specified transitions would not arise from a single episode.)

	$Q(D, \text{west})$	$Q(D, \text{east})$	$Q(E, \text{west})$	$Q(E, \text{east})$
Initial:	0	0	0	0
Transition 1: $(s = D, a = \text{east}, r = -1, s' = E)$		-0.5		
Transition 2: $(s = E, a = \text{east}, r = +2, s' = F)$				1.0
Transition 3: $(s = E, a = \text{west}, r = 0, s' = D)$				
Transition 4: $(s = D, a = \text{east}, r = -1, s' = E)$		-0.25		

The agent is still at the water park MDP, but now we're going to use function approximation to represent Q-values. Recall that a policy π is *greedy* with respect to a set of Q-values as long as $\forall a, s\ Q(s, \pi(s)) \geq Q(s, a)$ (so ties may be broken in any way).



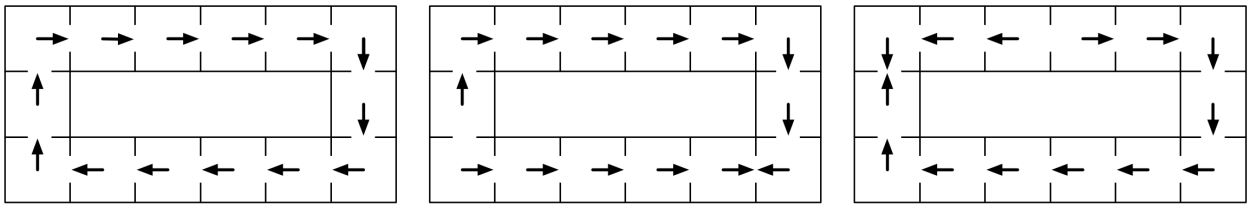
For the next subproblem, consider the following feature functions:

$$f(s,a) = \begin{cases} 1 & \text{if } a = \text{east}, \\ 0 & \text{otherwise.} \end{cases}$$

$$f'(s,a) = \begin{cases} 1 & \text{if } (a = \text{east}) \wedge \text{isSlide}(s), \\ 0 & \text{otherwise.} \end{cases}$$

(Note: isSlide(s) is true iff the state s is a slide square, i.e. either F or G.)

Also consider the following policies:



(d) Which are greedy policies with respect to the Q-value approximation function obtained by running the single Q-update for the transition $(s = F, a = \text{east}, r = +2, s' = G)$ while using the specified feature function? You may assume that all feature weights are zero before the update. Use discount $\gamma = 1.0$ and learning rate $\alpha = 1.0$. Circle all that apply.

f	π_1	$\boxed{\pi_2}$	π_3
f'	$\boxed{\pi_1}$	$\boxed{\pi_2}$	$\boxed{\pi_3}$

You see the sample (F, east, G, +2). Use approximate Q-Learning to update the weights.

You should get that the new weights are both going to be positive since the sample reward was positive and the feature value was on for both $f(F, east)$ [since you took action east] and $f'(F, east)$ [since you took action east, and you were on the water slide].

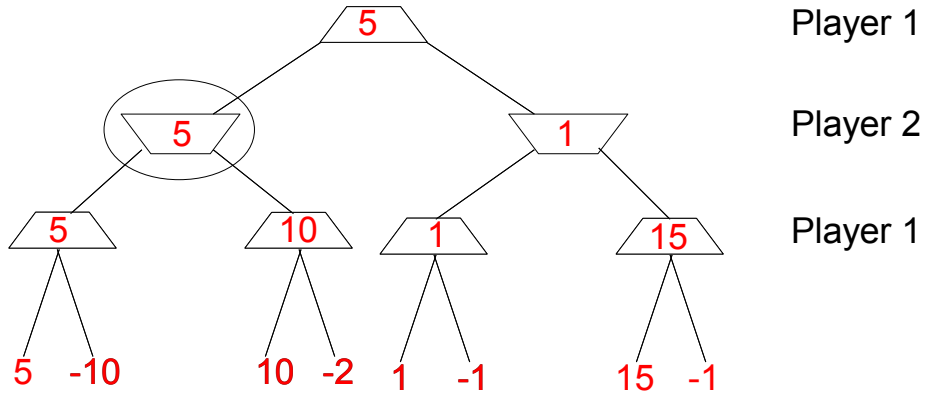
Now, with your new weights, you need to see which greedy policy can be possible.

For f, going East is preferred if possible (when you calculate the Q-value, any Q-state with action east has a positive value, anything else has a value of 0. Hence, throw out π_1 and π_3, since some arrows go west.

For f', going East is preferred *if* you are on the slide (otherwise, everything else is just 0). All three policies contain the fact that you move east from F and G, so all policies are good.

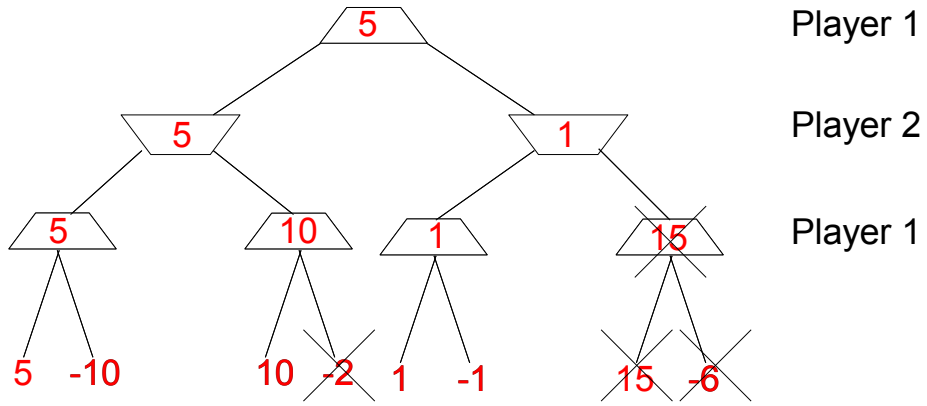
Q3. Minimax and Expectimax

(a) Consider the following zero-sum game with 2 players. At each leaf we have labeled the payoffs Player 1 receives. It is Player 1's turn to move. Assume both players play optimally at every time step (i.e. Player 1 seeks to maximize the payoff, while Player 2 seeks to minimize the payoff). Circle Player 1's optimal next move on the graph, and state the minimax value of the game. Show your work.

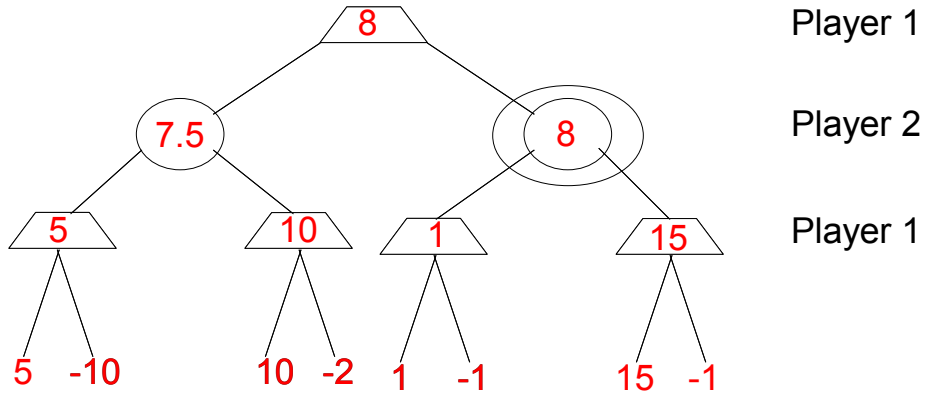


Player 1 should play Left for a payoff of 5.

(b) Consider the following game tree. Player 1 moves first, and attempts to maximize the expected payoff. Player 2 moves second, and attempts to minimize the expected payoff. Expand nodes left to right. Cross out nodes pruned by alpha-beta pruning.

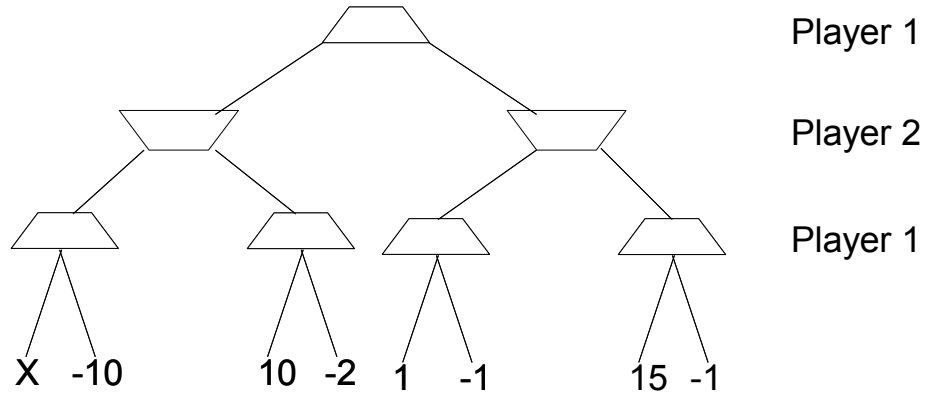


(c) Now assume that Player 2 chooses an action uniformly at random every turn (and Player 1 knows this). Player 1 still seeks to maximize her payoff. Circle Player 1's optimal next move, and give her expected payoff. Show your work.

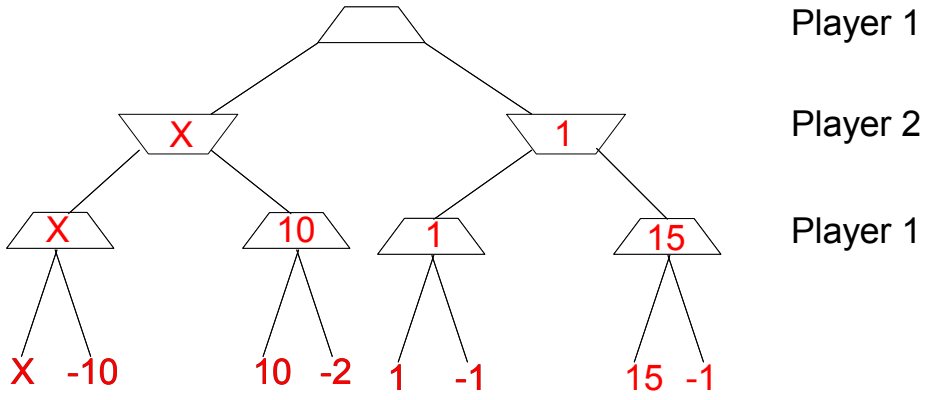


Player 1 should play Right for a payoff of 8.

Consider the following modified game tree, where one of the leaves has an unknown payoff x. Player 1 moves first, and attempts to maximize the value of the game.



(d) Assume Player 2 is a minimizing agent (and Player 1 knows this). For what values of x does Player 1 choose the left action?



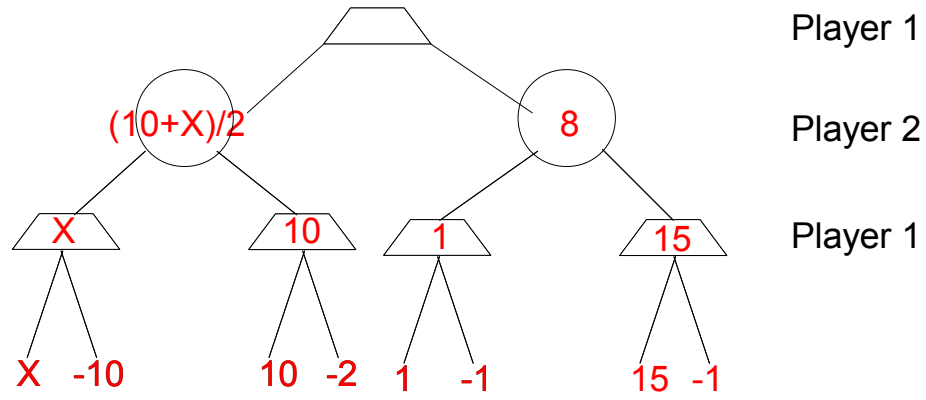
First, consider under which conditions the player will ultimately receive the x payout. The bottom left maximizing agent will choose the action leading to x if $x > -10$. The minimizing agent above it will choose the action leading to x as long as $x < 10$. The topmost maximizing agent will choose the action leading to x if $x > 1$.

So, for any $x > 1$, Player 1 will choose the left action (we also accepted $x \geq 1$).

Common mistakes:

- if you said $x \leq 10$. Even if $x > 10$, the optimal action is still to move left initially.
- for $x = 10$. The minimax value is equal to the expectimax value here, but we want the minimax value to be worth more than the expectimax value.

(e) Assume Player 2 chooses actions at random (and Player 1 knows this). For what values of x does Player 1 choose the left action?



Running the expectimax calculation we find that the left branch is worth $(10 + x)/2$ while the right branch is worth 8. Calculation shows that the left branch has higher payoff if $x > 6$. ($x \geq 6$ also acceptable)

(f) For what values of x is the minimax value of the tree worth more than the expectimax value of the tree?

The minimax value of the tree can never exceed the expectimax value of the tree, because the only chance nodes are min nodes. The value of the min node is (weakly) less than the value of the corresponding chance node, so the value the max player receives at the root is (weakly) less under minimax than expectimax.

Common mistakes:

- If you assume the minimax value of the tree is x for $x > 1$ and the expectimax value of the tree is $(x+10)/2$ for $x > 6$ and solve the inequality, you get $x > 10$ as the critical value for x, corresponding to a minimax payoff of x. However, the minimax value can not exceed 10, or the min player will choose the branch with value 10, so you cannot get payoffs $x > 10$.
- If you calculate the value of the left branch under minimax rules and the value of the right branch under expectimax, you may get $x > 8$.