CS 188
Spring 2019

Section Handout 3

The preamble is an abbrevation 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.

Policy Iteration

If all we want is to determine the optimal policy for the MDP value iteration tends to do a lot of overcomputation since the policy as computed by policy extraction generally converges significantly faster than the values themselves. This motivates **policy iteration**, an algorithm that maintains the optimality of value iteration while providing significant performance gains. It operates as follows:

1. Define an *initial policy.* This can be arbitrary, but policy iteration will converge faster the closer the initial policy is to the eventual optimal policy.

2. Repeat the following until convergence:

   - **Policy evaluation:** For a policy π, policy evaluation means computing $V^\pi(s)$ for all states s, where $V^\pi(s)$ is expected utility of starting in state s when following π:

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

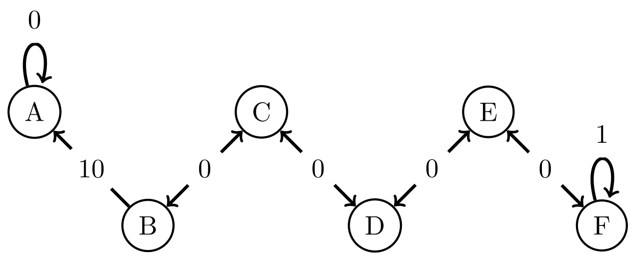
     Define the policy at iteration i of policy iteration as π_i. Since we are fixing a single action for each state, we no longer need the max operator which effectively leaves us with a system of $|S|$ equations generated by the above rule. Each $V^{\pi_i}(s)$ can then be computed by simply solving this system.

   - **Policy improvement:** Policy improvement uses policy extraction on the values of states generated by policy evaluation to generate this new and improved policy:

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$$

     If $\pi_{i+1} = \pi_i$, the algorithm has converged, and we can conclude that $\pi_{i+1} = \pi_i = \pi^*$.

1 MDP



Consider the MDP above, with states represented as nodes and transitions as edges between nodes. The rewards for the transitions are indicated by the numbers on the edges. For example, going from state B to state A gives a reward of 10, but going from state A to itself gives a reward of 0. Some transitions are not allowed, such as from state A to state B. Transitions are deterministic (if there is an edge between two states, the agent can choose to go from one to the other and will reach the other state with probability 1).

(a) For this part only, suppose that the max horizon length is 15. Write down the optimal action at each step if the discount factor is $\gamma = 1$.

(b) Now suppose that the horizon is infinite. For each state, does the optimal action depend on γ? If so, for each state, write an equation that would let you determine the value for γ at which the optimal action changes.

2 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?

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

States	0	2	3	4	5
V_0					
V_1					
V_2					
V_3					
V_4					

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

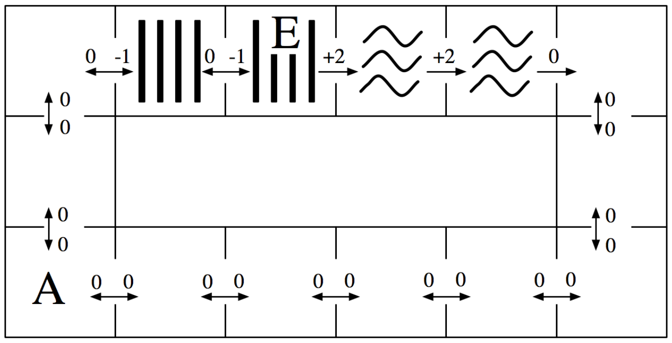
States	0	2	3	4	5
π^*					

(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}					
π_{i+1}					

Q3. 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?

(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		
$V_{10}^*(s)$	1.0		
$V_{10}^*(s)$	0.1		
$Q_1^*(s, \text{west})$	1.0	——	
$Q_{10}^*(s, \text{west})$	1.0	——	
$V^*(s)$	1.0		
$V^*(s)$	0.1		