Homework 3 of CS 165A (Spring 2023)

University of California, Santa Barbara

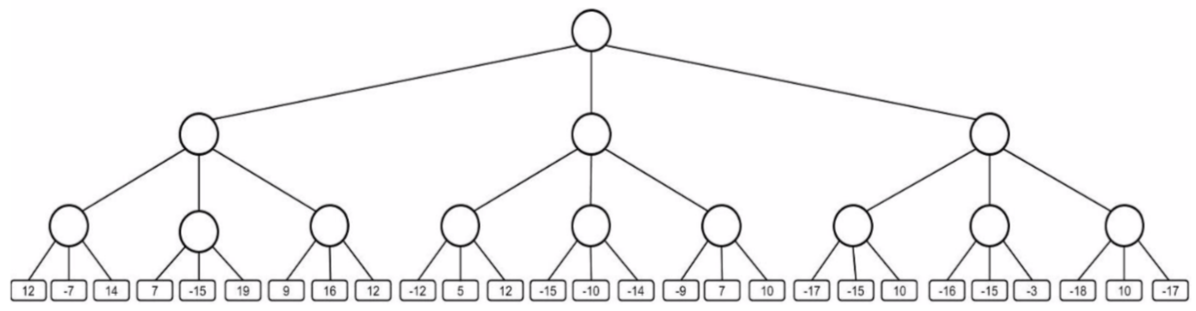
To be discussed on May 17, 2023 (Wednesday)

Notes:

- Please contact Vihaan Akshaay(`vihaanakshaay@ucsb.edu`) if you have any doubts regarding this homework.

Problem 1 Minimax Search, Pruning, and Expectimax

The following is a game-tree, whether the two players take turn to choose actions. You are the MAX player and you go first.



(a) Follow the Minimax algorithm and put a number in each circle in the following game tree in Figure ??.

(b) Assume that the order of expanding nodes is from left to right. Rerun the minimax algorithm with alpha-beta pruning in Figure ??

Indicate which nodes (subtrees) are pruned and the type of pruning (alpha or beta). Indicate the values of alpha and beta at each pruning step [1].

[1]The values of alpha and beta are the bounds based on the evidence that we have already collected

(c) Suppose that the adversary adopt an ϵ-greedy algorithm:

> With probability ϵ, the adversary choose a random action; and with probability $1 - \epsilon$, the adversary chooses the minimum.

Now you are asked to do an Expectimax search and mark the values of every node in Figure **??**, in which you need to maximize the expectation of reward over all next positions the adversary may choose (instead of the minimal reward as is in Minimax settings).

The parameter $0 \leq \epsilon \leq 1$. Write down the ranges of ϵ, for which the optimal first action of Player MAX will be LEFT, MIDDLE and RIGHT respectively.
Hint: you can make it in a similar way as Minimax search, by writing an expectation on each adversarial circle, and a maximum on each player's circle in Figure **??**.

Part(a): This tree contains three levels. The first and the third level are MAX agents and the second level is a MIN agent. For the MAX agent, the value is the maximum of all leaf nodes. For the MIN agent, the value is the minimum of its leaf nodes. Figure 1 has the MIN-MAX values for all nodes in the first figure.

Part(b): Video Solution. We repeatedly perform alpha-beta pruning by following the MAX_VALUE and MIN_VALUE functions given in figure 2.

Part(c): The adversary chooses an ϵ-Greedy algorithm. (ie. chooses a random action with probability ϵ and chooses the minimum value with probability 1-ϵ).

If we look at the first sub-tree, the MIN node chooses 14 with a probability (1-ϵ) as part of the exploitation phase. As part of the exploration phase, the agent chooses each of the three nodes with probability (ϵ/3). Therefore, we can say that the net value would be $(1 - \epsilon) \times 14 + (\epsilon/3) \times (14 + 19 + 16) = 14 + 7/3\epsilon$. Following the same method, we obtain values for all the MIN nodes. We can see that since ϵ always takes a value between 0 and 1, the first MIN node (first from left) will always have a higher value than the other two MIN nodes. Therefore, the main MAX node at the summit picks this value.

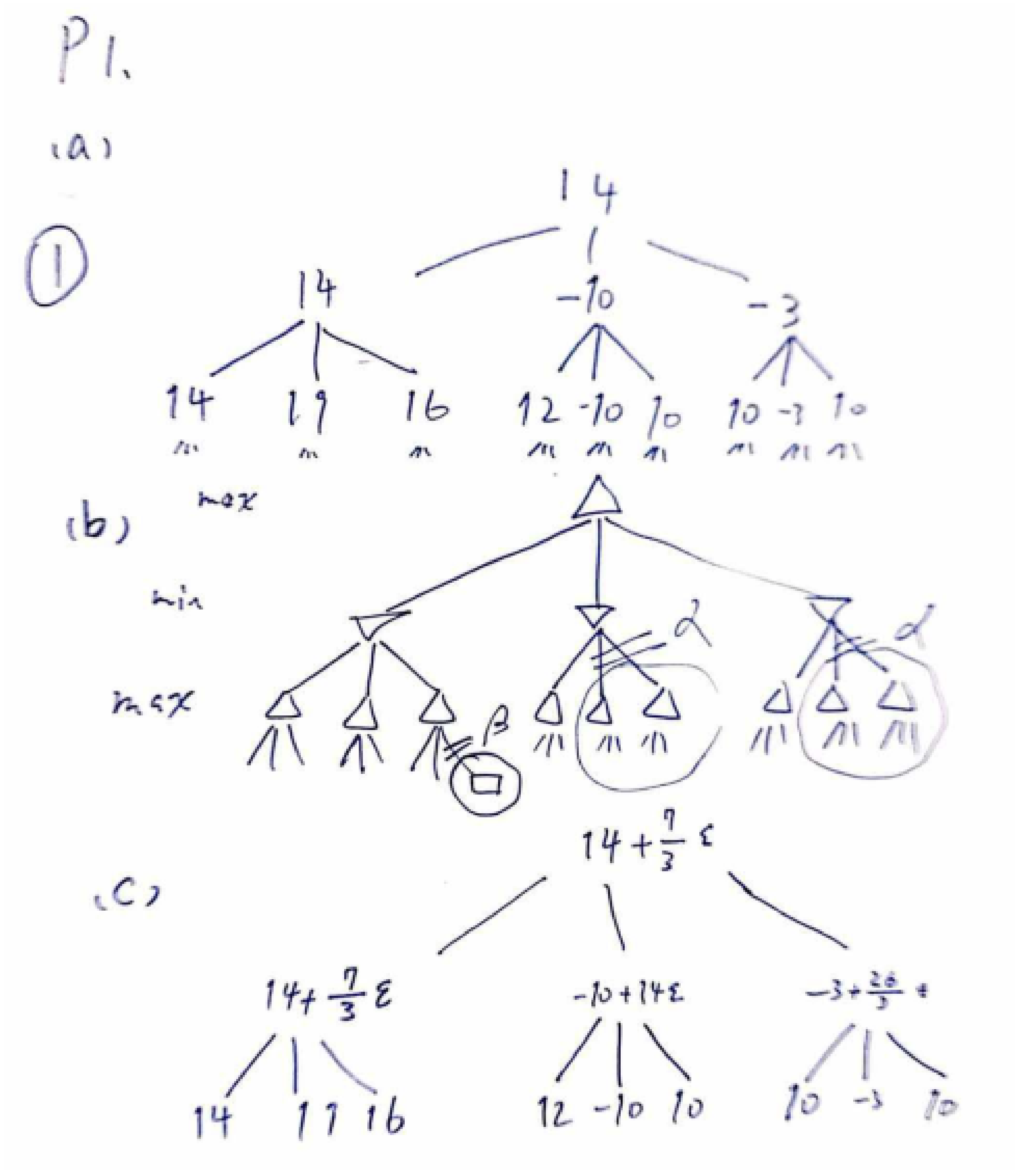


Figure 1: Problem 1 solution.

```
function MAX-VALUE($state, \alpha, \beta$) returns a utility value
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha$, $\beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$
```

```
function MIN-VALUE($state, \alpha, \beta$) returns a utility value
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow +\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Figure 2: MAX_VALUE and MIN_VALUE functions for Alpha-Beta Pruning

Problem 2 Tic-Tac-Toe with Evaluation functions

This problem exercises the basic concepts of game playing, using tic-tac-toe (noughts and crosses) as an example. We define X_n as the number of rows, columns, or diagonals with exactly n X's and no O's. Similarly, O_n is the number of rows, columns, or diagonals with just n O's. The utility functions assigns $+1$ to any state with $X_3 = 1$ and -1 to any state with $O_3 = 1$. All other terminal states have utility 0. For non-terminal states, we use a linear evaluation function defined as $Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$.

(a) What is the total number of states for the game of tic-tac-toe?

Solution: (a) (b) 3^9 is a crude answer to receive partial credits. Slightly better, you can have $1 + 9 + 9 * 8 + 9 * 8 * 7 + ... + 9 * 8 * 7 * ... * 1$, which takes in to account the rules of the game where people take turns. Even better, you can remove those states from each time there one of the players have already won. Here is a report about total number of possible states for curious readers.

(b) (Challenge question) How many possible game trees are there for tic-tac-toe? Include your workings towards an answer. If you write code to calculate the answer, please attach your code.

(b) The number if upper bounded by $9! = 362880$, i.e., each player take turns to fill out the 9 positions. A subset of these game trees will end early. People who answer 9! will get partial credits.

A more precise answer is the following: Calculate the number of ways the game in 1 ply, 2 plies, 3 plies, ..., 9 plies. Then just add them up, and you will get the exact answer. This report has information on total number of game trees as well.

(c) Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board). One way to simplify your game-tree is to take symmetry into account (namely, at the beginning there are essentially only three moves that player X can take; other moves will be equivalent to these three moves due to symmetry.).

(d) Mark on your tree the evaluations of all the states at depth 2.

(e) Using the minimax algorithm, mark on your tree the backed-up values for the states at depth 1 and 0, and use those values to choose the best starting move.

(f) Circle the nodes at depth 2 that would *not* be evaluated if alpha-beta pruning were applied, assuming the nodes are generated in the *optimal* order for alpha-beta pruning.

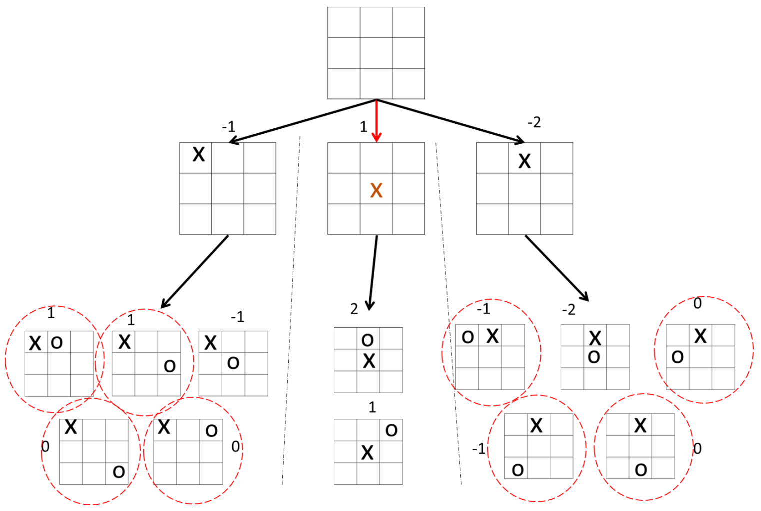


Figure 3: HW2 solution. Without losing generality, we let X go first. For (e), the middle(red) step is the optimal step for the former player. For (f), those circled in red are cases being pruned, and the prunings are all α- prunings.

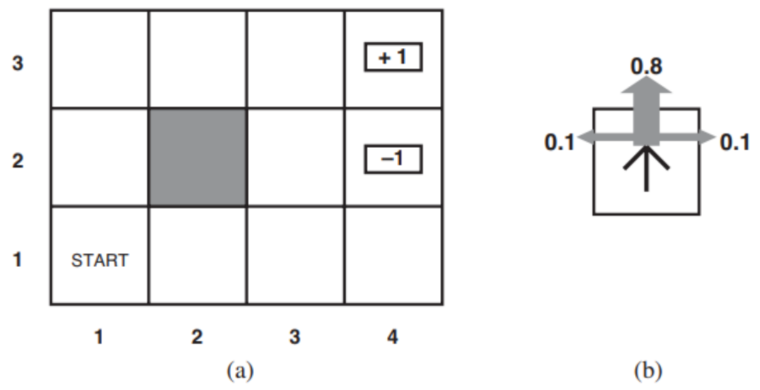


Figure 4: (a) A simple 4×3 environment that presents the agent with a sequential. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves towards one of the two directions perpendicular to the intended direction (0.1 for each such direction). A collision with a wall results in no movement (the agent stays where she is). When the agent transitions into the two terminal states a reward of +1 and –1 is received and the game ends. In addition, each unit time the agent spends in the environment costs 0.04 (a reward of -0.04 for every action taken).

Problem 3: MDP, Policy, V-function and Q-function in the 4×3 Grid World

In this problem, we will work with the the 4×3 grid world shown in Figure 4. This is the example we covered in the lecture. This question makes sure that you understand MDP.

The problem is best solved by **writing python codes**. You are feel free to structure your code in anyway you want. A good strategy would be to modularize your code by caching intermediate variables and writing functions. Representing the parameters of the MDP that you come up with in Part (a) as matrices and vectors is absolutely critical for a clean representation. So make sure you are representing the MDP parameters correctly (do dimension checks!) and can run the MDP as a simulator of the task environment.

(a) Recall that an MDP is determined by a tuple $(\mathcal{S}, \mathcal{A}, r, P, d_1, T)$, where $\mathcal{S}$ is the state-space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $P(s'|s, a)$ is the state-transition probability denoted by a conditional probability distribution[2], d_1 is the distribution of the initial state and T is the horizon of the MDP.

Read the description of the grid world environment in Figure 4 carefully and translate that description into an MDP. In particular, specify the all items in this tuple.

[2]This is a conditional probability table (CPT) when $\mathcal{S}$ and $\mathcal{A}$ are both discrete, for this homework we are only considering finite state spaces and action spaces, so we denote $S := |\mathcal{S}|$ and $A := |\mathcal{A}|$. Note that $\mathcal{S}$ and $\mathcal{A}$ are sets, so $| \cdot |$ with an input being a set outputs the *cardinality of a set*, or simply the number of items in a set.

(Hint 1: In the infinite horizon case $T = \infty$. You can think about hitting the terminal states the same as transitioning into a dummy sinking state $\perp$ with 0 rewards and it only transitions into itself. This is the case in this question.
Hint 2: In this example, r is a function of the state you transition into only, and you collect the reward / loss after the state-transition.
Hint 3: Do the dimension check for all elements in this tuple that describes an MDP. Use an extra piece of paper to carefully write down the matrices and vectors and fill in the numbers. This will help you understand MDP and help you with Q4 too.)

(b) Consider a *fixed* pre-defined sequence of actions [*Up, Up, Right, Right, Right*] (you take this sequence of actions without considering which state you are in.) Calculate which squares can be reached when you start from $(1, 1)$ and with what probabilities.

(Hint: the easiest way of calculating these probabilities will be to write a piece of code that manipulates your specified matrices and vectors from Part (a).)

(c) A **policy** is a "look-up table", or a "book of decision rules". Policy π determines at a given state s, which action $a \in \mathcal{A}$ to take. We use the notation $a = \pi(s)$ for deterministic policies. When π is stochastic, this book records the probability to taking each action given s, so a convenient notation to denote this policy is by the conditional distribution $\pi(a|s)$. A policy is called *stationary* if it does not depend on time t; otherwise it is called *non-stationary*, in which case we use π_t to denote the policy and it needs to be specified for all $t = 1, 2, ..., T$.

In this example, a deterministic policy can be described a figure with arrows as in those in Figure 5.

i. Can the above sequence of actions in Part (b) be generated by a *stationary* policy? If so, write π down. If not, write down a non-stationary policy π (consists of $\pi_1, ..., \pi_5$) that produces this sequence of actions.

ii. Let $T = 5$, calculate the value of this non-stationary policy π using your results in Part (b). Recall that the value of a policy in a finite horizon case (without discounting) is given by

$$v^\pi = \mathbb{E}_\pi \left[\sum_{t=1}^{T} R_t \right] = \sum_t \sum_s d_t^\pi(s) r_t^\pi(s)$$

where $d_t^\pi(s)$ denotes the probability of $S_t = s$ under policy π and

$$r_t^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] = \sum_a \pi_t(a|s) \sum_{s'} P(s'|s, a) r(s, a, s').$$

iii. Let $T = 5$, calculate the V_t^π function and the Q_t^π function for $t = 1, 2, 3, 4, 5$.
(Hint 1: Use the Bellman equations

$$V_t^\pi(s) = \sum_a \pi_t(a|s) \sum_{s'} P(s'|s, a)(r(s, a, s') + V_{t+1}^\pi(s'))$$

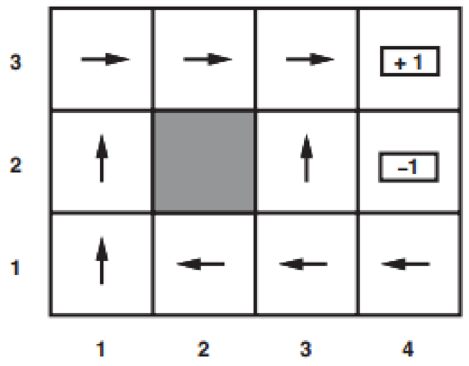


Figure 5: An optimal policy for the stochastic environment with $r(s, a, s') = -0.04$, $\gamma = 1$, $T = \infty$ when s' is a non-terminal states.

and

$$Q_t^\pi(s, a) = \sum_{s'} P(s'|s, a) \left(r(s, a, s') + \sum_{a'} \pi_{t+1}(a'|s') Q_{t+1}^\pi(s', a') \right).$$

Hint 2: Write a backup function that calculates V_t given V_{t+1}. Recursively call this function from $t = 5$ and go backwards. Define $V_6^\pi(\cdot) \equiv 0, Q_6(\cdot, \cdot) \equiv 0$.)

iv. Now let us use the alternative way of calculating the value of the policy π. $v^\pi = \sum_s d_1(s) V_1^\pi(s)$. Include your code and the output of the code. Verify that you calculations are correct by comparing this with Part (ii).

v. Now let us do a one step policy iteration. Define π' to be $\pi_t'(s) = \text{argmax}_a Q_t^\pi(s, a)$. Repeat your steps in Part iii and calculate $V_t^{\pi'}$ for $t = 1, 2, 3, 4, 5$. Is π' a better policy than π?

(d) Now let's move on to infinite horizon, discounted MDP. Take $T = \infty$. The main difference is that whenever the agent is at a state s, the future will be identical no matter what t it is that the agent is at. Therefore, the optimal policy will be stationary in this case.

i. The policy in Figure 5 is the optimal policy for this problem when $\gamma = 1$. Let us call it π^*. Now calculate the 5-step expected cumulative reward **by Monte Carlo** — obtaining many samples by repeatedly running this policy π^* starting from $(1, 1)$ for 5 steps. Then calculate the sample average of the cumulative rewards. Submit your code that implements it.

Yet another way of calculating this expression is to invoke your function from Part ii with $s = (1, 1)$, $\pi = \pi^*$ and $T = 5$. A good idea of testing that your solution is correct is to check whether you get the approximately the same numbers using these two approaches.)

ii. Write the python function that takes an initial state s, a policy π and the MDP as an input, then calculate the T-step cumulative reward using

$$\hat{V}^{\pi}(s) := \mathbb{E}_{\pi}\left[\sum_{t=1}^{T} R_t \middle| S_1 = s\right] = r^{\pi}(s) + \sum_{t=2}^{T}\sum_{s'} d_t^{\pi}(s'|S_1 = s) r^{\pi}(s').$$

where $r^{\pi}(s) := \sum_a \pi(a|s)\sum_{s'} P(s'|s,a)r(s,a,s')$, which measures the expected reward under s.

Take $T = 100$ will give a good approximation of $T = \infty$. Call the function with all s, print a table of the results.

(Hint: $d_t^{\pi}(s'|S_1 = s)$ denotes the probability of $S_t = s'$ when $S_1 = s$ when all actions are taken by policy π. Think about a $t-1$-step random walk starting at state s, what is the transition matrix under π?)

iii. Write a function to calculate the scalar version of Bellman error. The function takes a vector $\hat{V} \in \mathbb{R}^S$, γ and the MDP as an input and outputs a scalar

$$\sqrt{\sum_s \left(\hat{V}(s) - \max_a \sum_{s'} P(s'|s,a)(r(s,a,s') + \gamma \hat{V}(s'))\right)^2}.$$

If the Bellman error is 0, then we know that $\hat{V} = V^*$.

Substitute $\hat{V}^{\pi^*}$ that you obtained in Part ii into this function with γ chosen to be 1 and calculate the Bellman error. Is this close to 0?

Submit your output, the python function and the script that runs this function.

iv. So far, we are only validating that the policy in Figure 5 is optimal. Write a python function that takes **value iterations**:

$$V_{i+1}(s) \leftarrow \max_a \left\{\sum_{s'} P(s'|s,a)(r(s,a,s') + \gamma V_i(s'))\right\}.$$

The function takes an arbitrary vector $V_0 \in \mathbb{R}^S$, integer k, and the MDP as inputs, then it runs value iterations for k iterations, then output $V_k \in \mathbb{R}^S$. Take $\gamma = 0.9$, plot the Bellman error as a function of k.

v. Write down the formula to obtain the optimal policy π^* using V^* associated with a discounting factor of γ. Use this formula to derive a policy π^k by plugging V_k (from Part iv) in place of V^* the policy that comes from the previous question.

- Take $\gamma = 0.9, k = 100$ and print out the policy π^k. Is the policy same as the one in Figure 5?
- How about when $\gamma = 0.5$, $k = 100$?

Solution:

(a) $S = \{(1,1),(1,2),\ldots,(4,1),(4,2),(4,3)\} \triangleq [s_1, s_2, \ldots, s_{11}]^T$

In the code that is shared, we also use another notation that assigns a number to these states from 0-11.

A: $\{"up", "down", "left", "right"\}$

$$r(s,a,s') = r(s') \begin{cases} 0.96 \; elif s' == (4,3) \\ -1.04 \; elif s' == (4,2) \\ -0.04 \; else \end{cases}$$

$P(s'|s,a)$ can be written as 4 11×11 matrices: $P_{up} = P(\cdot|\cdot, "up"), P_{right} = P(\cdot|\cdot, "right"), P_{left} = P(\cdot|\cdot, "left"), P_{down} = P(\cdot|\cdot, "down")$.

$d_1 = [0,0,0,0,0,0,0,0,1,0,0,0]^T$ (Since the start state is the 8th element in the above formulation)

T is the time horizon, and it could be infinite (but not necessarily).

(b) $d_6 = (P_{right})^3 (P_{up})^2 d_1$

(c) (i) This is not a stationary policy. Now let's write a non-stationary policy representing this: $\pi_1(s) = \pi_2(s) = \text{"up"}, \forall s \; \pi_3(s) = \pi_4(s) = \pi_5(s) = \text{"right"}, \forall s$

(ii) Here

$$r_t^\pi(s) = \sum_a \pi_t(a|s) \cdot \sum_{s'} P(s'|s,a) r(s,a,s')$$
$$= \begin{cases} \sum_{s'} P_{up_{s',s}} \cdot r(s,s') \; , when \; t \; = \; 1, \; 2 \\ \sum_{s'} P_{right_{s',s}} \cdot r(s,s') \; , when \; t \; = \; 3, \; 4, \; 5 \end{cases}$$

and then :

$$v^\pi = \sum_t (d_t^\pi)^T (r_t^\pi)$$

where d_t^π is the vector comes from the derivation of part (b).

(iii) For value function, we have:

$$V_5^\pi = \sum_{s'} (P_{right})_{s',s} \cdot r(s,s')$$

$$V_4^\pi = \sum_{s'} (P_{right})_{s',s} \cdot r(s,s') + (\mathbf{V_5^\pi} P_{right})(s)$$

$$V_3^\pi = \sum_{s'} (P_{right})_{s',s} \cdot r(s,s') + (\mathbf{V_4^\pi} P_{right})(s)$$

$$V_2^\pi = \sum_{s'} (P_{up})_{s',s} \cdot r(s,s') + (\mathbf{V_3^\pi} P_{up})(s)$$

$$V_1^\pi = \sum_{s'} (P_{up})_{s',s} \cdot r(s, s') + (\mathbf{V_2^\pi} P_{up})$$

. For Q-function, we have:

$$Q_5(s,a) = \sum_{s'} (P_{"a"})_{s',s} \cdot r_{s,s'}$$

$$Q_4(s,a) = \sum_{s'} (P_{"a"})_{s',s} \cdot (r_{s,s'} + Q_5(s', "right"))$$

$$Q_3(s,a) = \sum_{s'} (P_{"a"})_{s',s} \cdot (r_{s,s'} + Q_4(s', "right"))$$

$$Q_2(s,a) = \sum_{s'} (P_{"a"})_{s',s} \cdot (r_{s,s'} + Q_3(s', "right"))$$

$$Q_1(s,a) = \sum_{s'} (P_{"a"})_{s',s} \cdot (r_{s,s'} + Q_2(s', "up"))$$

(iv) $v^\pi = (d_1)^T V_1^\pi = V_1^\pi(1)$

(v) Demo:

Step 1: compute $\pi_t^{'}(s)$ for $t = 1, 2, 3, 4, 5$. Do it element-wise.

Step 2: compute $V_t^\pi(s)$ for $t = 5, 4, 3, 2, 1$.

(d) (i)For Monte-Carlo simulations, we repeatedly start the agent from the start state and run the environment for several episodes. As this is run, we calculate the cumulative reward for each of these episodes and return the average.

(ii),(iii),(iv) Please check the codes that was shared.

(v)

$$\pi^*(s) = \arg\max_a \left\{ \sum_s P(s'|s,a) \left[r + \gamma V^*(s') \right] \right\}$$

Problem 4: MDPs and Value Iterations (Challenge problem, try this if you's like a bit more math)

Let $V_i \in \mathbb{R}^S$ be the value function estimate for all the states at i^{th} iteration. A **Bellman update** used in the Value Iteration is the following[3]:

$$V_{i+1}(s) \leftarrow \max_a \left\{ r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_i(s') \right\}.$$

where $0 < \gamma < 1$.

If we note this update as an operator $\mathcal{B}$, then we have:

$$V_{i+1} \leftarrow \mathcal{B}\ V_i.$$

Notice that the operation $\mathcal{B}\ V$ is **not** a matrix-to-vector product. Now we are going to prove that the Bellman operator B is a contraction.

(a) Show that, for any functions f and g,

$$|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$$

(Hint: Discuss the two cases of the absolute value on the LHS. Also, it might be helpful to define $\tilde{a} := \text{argmax}_a f(a)$ and $\tilde{a}' := \text{argmax}_a g(a)$. Similar to Q5 in HW1, a good way of thinking about $\max_a f(a)$ is that it is larger than $f(a')$ for any a'.)

(b) Substitute the definition of the Bellman operator into $|\mathcal{B}\ V_i(s) - \mathcal{B}\ V_i^{'}(s)|$, such that you can apply your result in Part (a).

(c) A **max norm** is defined as follows:

$$\|V\|_{\max} := \max_{s \in \{1,2,3,\ldots,S\}} |V(s)|.$$

Prove that for any two value function estimates $V, V' \in \mathbb{R}^S$, we have:

$$\|\mathcal{B}\ V - \mathcal{B}\ V'\|_{\max} \leq \gamma \|V - V'\|_{\max}.$$

(Hint: Apply the result in Part (a) to what you get in Part (b). Explicitly state how you are instantiating the result in Part (a) — what is f and what is g here.)

(d) Use the result in Part (c) to prove that the value iteration *works*, namely, as $i \to \infty$, $V_i \to V^*$.

[3]When the reward function depends on s, a, s' as in the lecture, then $r(s,a) = \sum_{s'} P(s'|s,a) r(s,a,s')$. Check that this is the same as the value iteration updates we define in Slide 11 of Lecture 13

Solution: (a) Proof: Suppose:

$$\arg\max_a f(a) = a_1$$

$$\arg\max_a g(a) = a_2$$

$$\arg\max_a |f(a) - g(a)| = a_3$$

Then, $LHS = |f(a_1) - g(a_2)|$. Without losing generality, suppose:

$$\max f(a) \geq \max g(a)$$

, then

$$LHS = f(a_1) - g(a_2)$$

. Since $g(a_2) \geq g(a), \forall a \in \mathbf{D}$, we have:

$$LHS = f(a_1) - g(a_2)$$

$$\leq f(a_1) - g(a_1)$$

$$\leq |f(a_1) - g(a_1)|$$

$$\leq \max |f(a) - g(a)|$$

■

(b)

$$|B\ V_i(s) - B\ V_i^{'}(s)|$$

$$= |\max_a r(s,a) + \gamma \sum_{s'} P(s'|a,s)V_i(s') - \max_a r(s,a) + \gamma \sum_{s'} P(s'|a,s)V_i^{'}(s')|$$

(c)

$$LHS = \max_s |\max_a r(s,a) + \gamma \sum_{s'} P(s'|a,s)V_i(s') - \max_a r(s,a) + \gamma \sum_{s'} P(s'|a,s)V_i^{'}(s)|$$

According to the conclusion in part(a), we have:

$$\leq \max_s \max_a |r(s,a) + \gamma \sum_{s'} P(s'|a,s)V_i(s') - (r(s,a) + \gamma \sum_{s'} P(s'|a,s)V_i^{'}(s))|$$

$$= \max_s \max_a |\gamma \sum_{s'} P(s'|a,s)(V_i(s') - V_i^{'}(s'))|$$

Since $P(s'|s,a) \geq 0$, we have the followings by applying "Inequality of absolute values":

$$LHS \leq \max_s \max_a (\gamma \sum_{s'} P(s'|s,a)|V_i(s') - V_i^{'}(s')|)$$

$$\leq \max_s \max_a \gamma \sum_{s'} P(s'|s,a) \cdot \max_{s''} |V_i^{'}(s'') - V_i(s'')|$$

$$= \max_s \max_a \gamma \sum_{s'} P(s'|s,a)||V' - V|| = \gamma \cdot ||V' - V||$$

(d) Let the initial values be V^0 and the consequent values after performing value iteration for i steps be V^i (ie $B^i\ V^0 = V^i$). Using the inequality above, we have the following:

$$||B\ V^i - B\ V^{i-1}||_{\max} \leq \gamma ||V^i - V^{i-1}||_{\max}$$

Substituting, the bellman operator applied values with the next iteration values:

$$||V^{i+1} - V^i||_{\max} \leq \gamma ||V^i - V^{i-1}||_{\max}$$

As we extend this until we reach the initial estimate, we obtain the following:

$$||V^{i+1} - V^i||_{\max} \leq \gamma^i ||V^1 - V^0||_{\max}$$

We know $B\ V^* = V^*$ and $\gamma \in (0,1)$. From the inequality above, as $i \to \infty$, the RHS of the equation tends to zero.

$$||B\ V^i - V^i||_{\max} \to 0 \implies V^i \to V^*$$