Homework 3 of CS 165A (Spring 2022)

University of California, Santa Barbara

To be discussed on May 13, 2022 (Friday)

Notes:

- The homework is optional. You do not need to submit your solutions anywhere and you will not be evaluated by these.
- To maximize your learning, you should try understanding the problems and try solving them as much as you can before the discussion class.
- Feel free to discuss with your peers / form small groups to solve these problems.
- Feel free to discuss any questions with the instructor and the TA in office hours or on Piazza.

Why should I do this homework?

This optional homework offers simple practices to help you understand how Game solving agents work (Problem 1 and 2), and for you to understand Markov decision processes (MDPs) (Problem 3 and 4). In particular, Problem 3 is a (somewhat tedious) hands-on practice that are best done by writing Python / numpy codes interactively, e.g., in a Jupyter notebook environments. It helps you to understand MDPs and gain further insight into value functions and Bellman equations. Problem 4 helps you to understand how "value iteration" works with a few simple mathematical derivations.

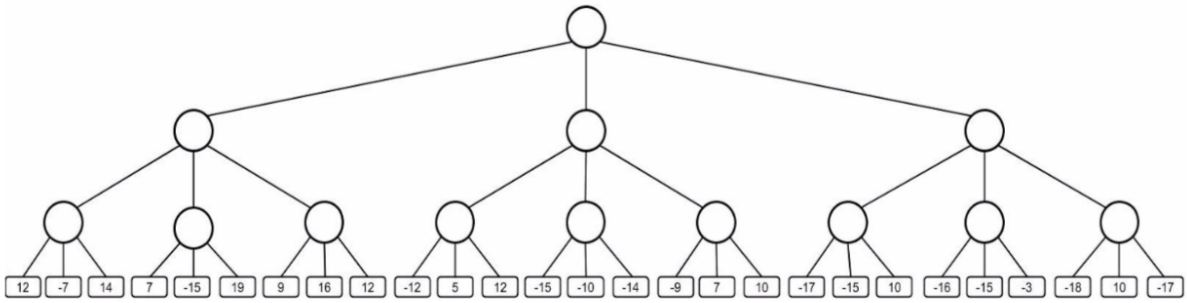
Why study game AI? Because games are fun! Being able to think strategically in complex game plays (sometimes as a simulation of real life) is what makes us so different from other animals. Solving games with minimax search and an evaluation function (a heuristic function) is a simple but powerful technique and arguably used implicitly by human game players. You will probably be able to beat the best chess player in your household with relatively simple heuristics and searching for only a handful of plies. Expectimax also offers a bridge between search and MDPs (try to figure out the precise connection and differences between two seemingly related things is a great way of learning known as "Near-Miss").

Why study MDPs? MDP is the main mathematical model we consider when studying Reinforcement Learning, which would be the single most important topic that we will spend many lectures on during the second half of the course. MDPs can be used to model discrete (and continuous) controlled processes such as robot navigation, chemical reactions, queuing systems, inventory managements, epidemic control and even for designing macro-economic policies. Understanding the basics of MDPs will help you to spot new applications whenever MDP is applicable. More pragmatically, if you still struggle with solving coding questions involving dynamic programming algorithms (e.g., variants of knapsack problems), you will be surprised to know how many of them are special cases of value iterations and MDPs. Learning MDPs well will help to to master the a big hammer for cracking tech interview questions (if you are graduating soon).

Finally, attempting these problems and showing up in the recitations may help you with Coding Project 2, 3 and the final exam.

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 1.

(b) Assume that the order of expanding nodes is from left to right. Rerun the minimax algorithm with alpha-beta pruning in Figure 2

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].

(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.

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

Now you are asked to do an Expectimax search and mark the values of every node in Figure 3, 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 3.

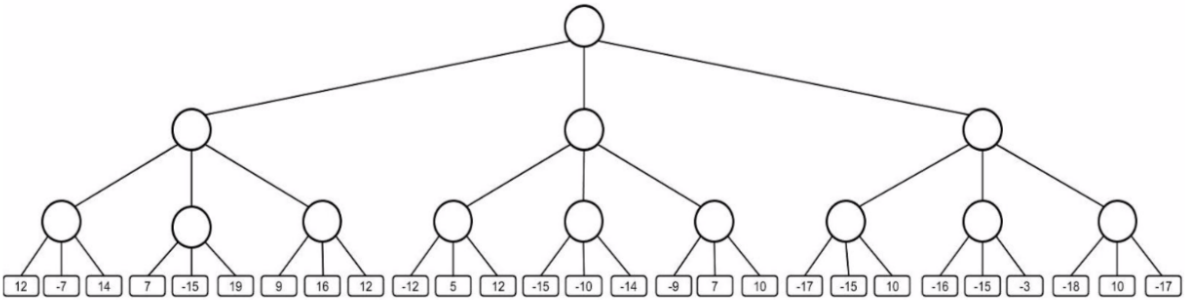


Figure 1: A game tree. [Provide your marked answers for Q1(a) in this figure.]

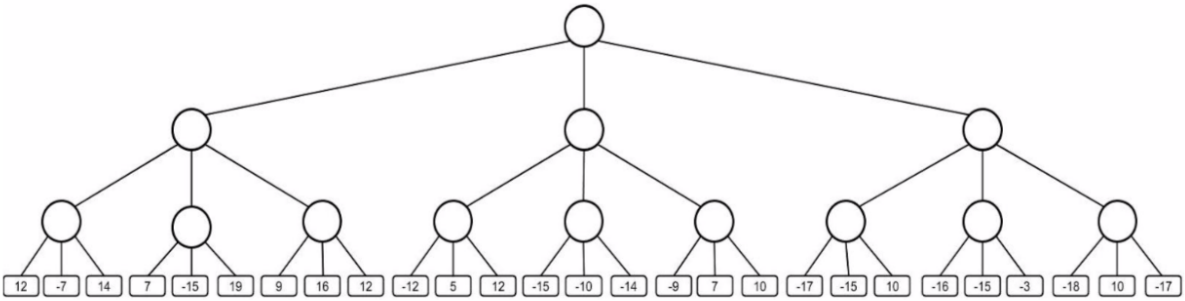


Figure 2: A game tree. [Provide your marked answers for Q1(b) in this figure.]

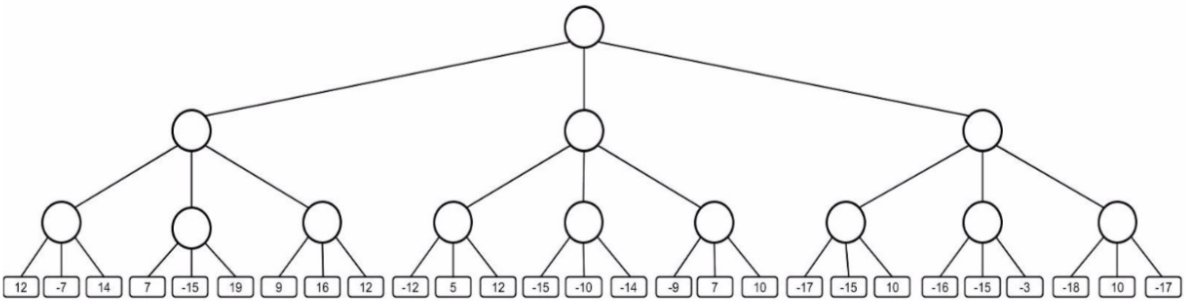


Figure 3: A game tree. [Provide your marked answers for Q1(c) in this figure.]

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?

(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.

(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.

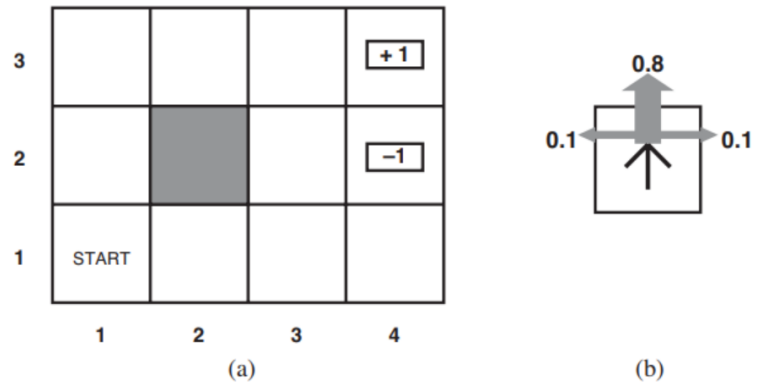


Figure 4: (a) A simple 4×3 environment that presents the agent with a sequential. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves towards one of the two directions perpendicular to the intended direction (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

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

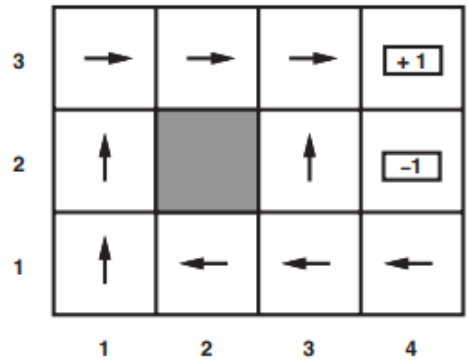


Figure 5: An optimal policy for the stochastic environment with $r(s, a, s') = -0.04$, $\gamma = 1$, $T = \infty$ when s' is a non-terminal states.

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 $\gamma = 0.5$, $k = 100$?

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