Practice Exam F

1 MDPs

1. (20 points) After your finals, you are relaxing on the couch watching TV. There is a remote control that you can reach while sitting on the couch to turn on the TV. Unfortunately, both the TV and the remote are buggy: the TV turns off randomly, and the clicking the remote often has no effect. You can also get up off the couch and turn on the TV manually.

   Let's formulate this scenario as an infinite-horizon MDP. To start:

   - $\mathcal{S} = \{\texttt{tvOn}, \texttt{tvOff}\}$
   - $\mathcal{A} = \{\texttt{clickRemote}, \texttt{pressTV}, \texttt{doNothing}\}$
   - $\gamma = 0.9$

   The transition distribution $P(s' \mid s, a)$ is specified as follows:

   - If $a = \texttt{clickRemote}$, the TV switches between on and off with 0.4 probability, and stays the same with 0.6 probability.
   - If $a = \texttt{pressTV}$, the TV switches between on and off with 1.0 probability.
   - If $a = \texttt{doNothing}$ and the TV is on, it switches off with 0.05 probability.
   - If $a = \texttt{doNothing}$ and the TV is off, it stays off with 1.0 probability.

   The reward function $R(s, a, s')$ is specified as follows, starting at 0:

   - If $s' = \texttt{tvOn}$, $+3$ is added to the reward.
   - If $a = \texttt{clickRemote}$, -1 is added to the reward.
   - If $a = \texttt{pressTV}$, -2 is added to the reward.

   You are considering the following policy:

   - $\pi(\texttt{tvOn}) = \texttt{doNothing}$
   - $\pi(\texttt{tvOff}) = \texttt{clickRemote}$

   (a) Write a system of linear equations to evaluate (compute the value function for) π. You do not need to solve the system.

   > **Solution:** $V(\texttt{tvOn}) = (0.05 * 0.9)V(\texttt{tvOff}) + 0.95(3 + 0.9V(\texttt{tvOn}))$
   > $V(\texttt{tvOff}) = 0.6(-1 + 0.9V(\texttt{tvOff})) + 0.4(2 + 0.9V(\texttt{tvOn}))$

   This policy seems reasonable, but having just aced an exam that covered MDPs, you are determined to compute a good policy in a principled way.

   In the course of performing value iteration, you have the following optimal value estimates:

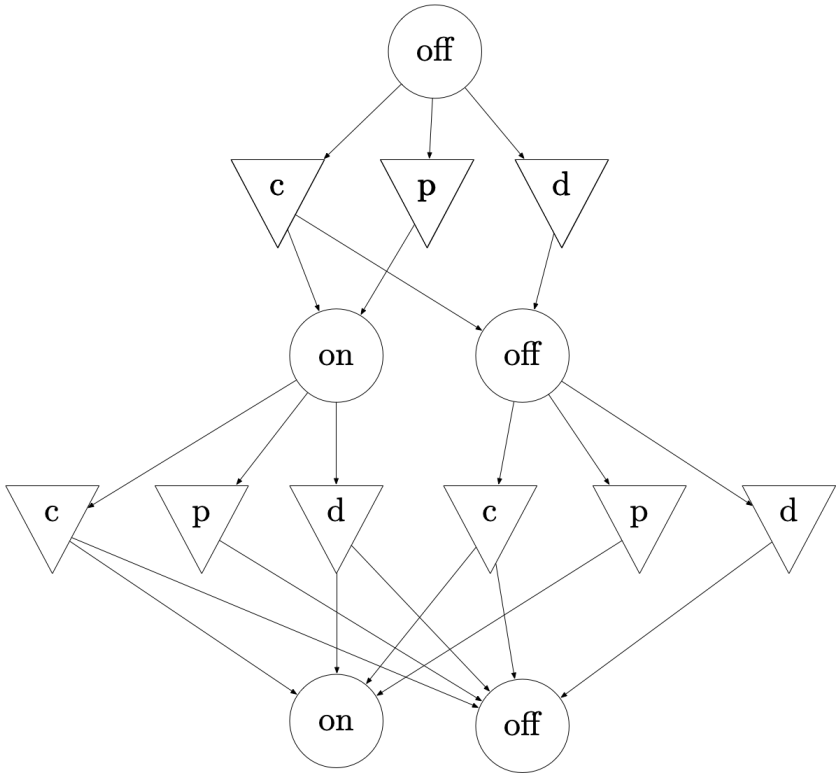
   - $V^*(\texttt{tvOn}) = 2$
   - $V^*(\texttt{tvOff}) = 1$

   (b) What would the new optimal value estimate for $V^*(\texttt{tvOn})$ be after one more iteration of value iteration?

Solution:
$$V^*(\texttt{tvOn}) \leftarrow \max \begin{cases} 0.4(-1+0.9V^*(\texttt{tvOff})) + 0.6(2+0.9V^*(\texttt{tvOn})) & [\texttt{clickRemote}] \\ -2+0.9V^*(\texttt{tvOff}) & [\texttt{pressTV}] \\ 0.05(0.9V^*(\texttt{tvOff})) + 0.95(3+0.9V^*(\texttt{tvOn})) & [\texttt{doNothing}] \end{cases}$$
$$V^*(\texttt{tvOn}) \leftarrow \max \begin{cases} 0.4(-1+0.9) + 0.6(2+1.8) & [\texttt{clickRemote}] \\ -2+0.9 & [\texttt{pressTV}] \\ 0.05(0.9) + 0.95(3+1.8) & [\texttt{doNothing}] \end{cases}$$
$$V^*(\texttt{tvOn}) \leftarrow 4.605$$

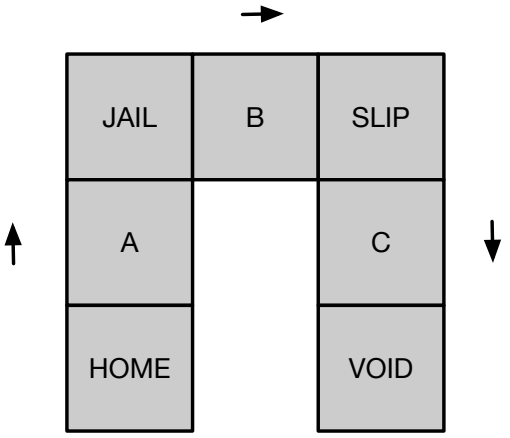
(c) Unfortunately, all good TV watching sessions must come to an end. Let us now consider a finite-horizon version of the above MDP. Everything remains the same, except now $H = 2$ (so we will take two actions) and $\gamma = 1$. We also know that the TV is off in the initial state. We will want to perform expectimax search to compute an optimal partial policy. Show the full AODAG *without any value annotations*. Do not do any calculations here!

Solution:



2 Monotony

Here's a terrible board game.



- You start in **HOME** and the game ends when you reach **VOID**.
- It is very boring and you have no choices to make.
- You move clockwise around the board. In **HOME** and the squares with arrows, you always just move to the next square.
- In **JAIL**, with probability .5 you stay in **JAIL** and with probability .5 you move to **B**.
- In **SLIP**, with probability .5 you go to **HOME** and with probability .5 you move to **C**.
- You get reward -1 on every step until your miserable existence ends in **VOID**.

(a) Write a system of 4 equations in 4 unknowns relating the undiscounted values of **HOME**, **JAIL**, **SLIP** and **VOID**.

Solution:

$$\begin{aligned} V_H &= -2 + V_J \\ V_J &= .5(-1 + V_S) + .5V_J \\ V_S &= .5(-1 + V_V) + .5V_H \\ V_V &= 0 \end{aligned}$$

(b) Now, imagine that you can choose actions!

- In any state, you can select the *escape* action at a cost of 3, or do the normal action described above (which we will call *go*).
- In the **JAIL** state, if you select the *escape* then you move to the right with probability 1. in the **SLIP** state, if you *escape*, then you move down with probability 1.

Compute 7 iterations if value iteration, showing the Q values for the *go* action in all states and for the *escape* action in **JAIL** and **SLIP**.

Solution:

Q(H, G)	Q(A, G)	Q(J, G)	Q(J, E)	Q(B, G)	Q(S, G)	Q(S, E)	Q(C, G)
-1	-1	-1	-3	-1	-1	-3	-1
-2	-2	-2	-4	-2	-2	-4	-1
-3	-3	-3	-5	-3	-2.5	-4	-1
-4	-4	-4	-6	-3.5	-3.0	-4	-1
-5	-5	-4.75	-6.5	-4.0	-3.5	-4	-1
-6	-5.75	-5.375	-7.0	-4.5	-4.0	-4	-1

```
-6.75   -6.375   -5.9375 -7.5    -5.0    -4.5    -4      -1
-7.375  -6.9375  -6.4687 -8.0    -5.0    -4.875  -4      -1
-7.9375 -7.46875 -6.7344 -8.0    -5.0    -5.1875 -4      -1
-8.4687 -7.73437 -6.8672 -8.0    -5.0    -5.4687 -4
-8.7344 -7.86719 -6.9336 -8.0    -5.0    -5.7343 -4
```

(c) What is the optimal policy?

Solution: Escape from slip, but just Go in Jail.

(d) Time for capitalism! Instead of *escaping* the void, we will try to make money. The squares A, B, and C start out as empty lots, but if you visit one and it's empty, you can do the action *invest*, which has reward -5, and changes the state of that square to a rental property. If you land on a square with a rental property, you get reward +10. All other actions have reward 0.

i. Is it necessary to add discounting to this problem for the values to be well defined? Why or why not?

Solution: No, because you end up in the void with probability 1 and the game ends, so all values are finite even without discounting.

ii. What is the state space for this new problem?

Solution: Home, A-Empty, A-Rental, Jail, B-Empty, B-Rental, Slip, C-Empty, C-Rental, Void.

3 Back it up!

2. Here is (incomplete) code for a dynamic programming algorithm

```
def run_dp(MDP):
  for all a in MDP.actions: Q[a] = fit()
  X = uniform random sample of N points from MDP.states
  while not done(<TODO>):
    Y = [backup(Q, s, a) for s in X]
    Q_new[a] = fit(X, Y)
    Q = Q_new
  return Q
```

Here, `fit()` is a random initialization for a function approximator, and `fit(X, Y)` is a function approximation procedure that learns a mapping between `X` and `Y`.

We wish to run this code on an MDP with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{P}$ where $\mathcal{P}$ is a probability or probability density value, and reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

(a) (2 points) What are the consequences of initializing randomly inside `fit()` instead of forcing it to output 0 initially?

Solution: Q-learning will still always work (importantly, because we are sampling X uniformly at random from S, if we sampled from the policy, this wouldn't be true); convergence speed might be affected.

(b) (2 points) Recall that the exact version of the backup operation is

$$\texttt{backup}(Q, s, a) = r(s, a) + \gamma \, \mathbb{E}_{s' \sim T(s,a,\cdot)} \left[\max_{a' \in A} Q(s', a') \right]$$

Explain concretely what would prevent us from computing this exactly when our transition model is implemented as a simulator?

> **Solution:** We need to be able to take an expectation over next states via the transition function. However, most simulators do not give access to transition probabilities directly. Another fair answer: the simulator needs to be resettable to an arbitrary state.

(c) (2 points) Write (in words or informal pseudocode) a function that implements a policy, given the output of `run_dp`, for a discrete action space. You can use $Q(s, a)$ to denote calling the resulting Q function from `run_dp` on a new state s and action a.

> **Solution:**
> ```
> def policy(s):
>   return argmax_a Q(s, a)
> ```

(d) (2 points) Explain why, when S is continuous, we could not use a table as a function approximator.

> **Solution:** We are extremely unlikely to encounter the same state twice. Thus, we will never quite learn anything and the Q-function will never improve significantly.

(e) (2 points) If $\mathcal{A}$ is continuous, then we cannot use separate Q-functions indexed by a (as in the above code). One solution could be to learn a single neural network that takes a state and action as input and outputs a Q-value (i.e., $Q(s, a)$). What is one disadvantage of this strategy?

> **Solution:** Optimization over Q-values is required to create a policy at performance time. Neural networks are often overconfident for values they haven't seen before: thus, trying to find a high Q-value at performance time could lead to selecting an action that hasn't been seen much at training3 time, but actually does not perform that well.

4 Truth in advertising

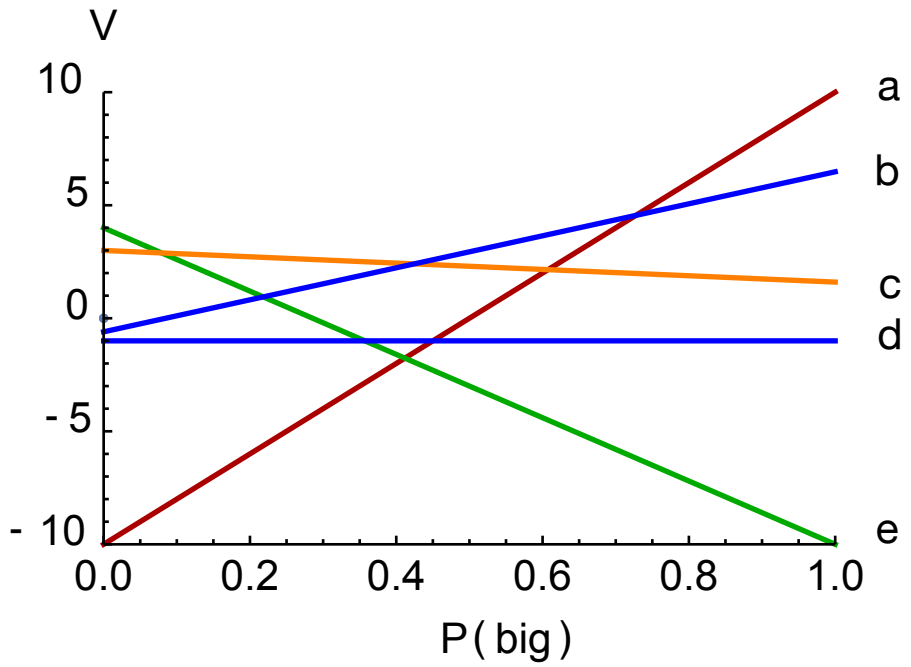
You are selling beautiful river stones as door-stops. Your e-commerce software is bad, though, and only lets you sell two different products, a *large stone* and a *small stone*. You are in a hurry because of the holiday season, and you need to go through your stone collection, decide how to label each one, and package them up for shipping.

You will model the process, for a single stone, as a POMDP, with

- States: {large-stone, small-stone}
- Actions:
  - ship-large: label this stone as a large stone and ship it
  - ship-small: label this stone as a small stone and ship it
  - make-small: grind the stone down to try to make it smaller

  - inspect: measure the stone to determine its size
- Transitions:
  - inspect does not change the state of the rock
  - ship-large and ship-small do not change the state of the rock, but do terminate the game
  - make-small: if the rock was originally large, then it becomes small with probability 0.9 and stays large with probability 0.1; if it was originally small, it stays small.
- Observations
  - There are three possible observations: looks-big, looks-small, and hazy.
  - All actions besides inspect generate observation hazy with probability 1.
  - The inspect action, on a large rock generates observations with distribution: {looks-big : 0.7, looks-small : 0.1, hazy : 0.2}
  - The inspect action, on a small rock generates observations with distribution: {looks-big : 0.2, looks-small : 0.5, hazy : 0.3}
- Rewards
  - Rewards for actions inspect and make-small are -1 in all states
  - Reward for action ship-small is -10 if the rock is large and $+4$ if the rock is actually small.
  - Reward for action ship-big is $+10$ if the rock is large and -10 if the rock is actually small.

This figure indicates α vectors for several simple policy trees.



For each policy tree below, indicate which vector it corresponds to:

(a) ship-large √ **a** ◯ b ◯ c ◯ d ◯ e

(b) ship-small ◯ a ◯ b ◯ c ◯ d √ **e**

(c) inspect ◯ a ◯ b ◯ c √ **d** ◯ e

(d) make-small then ship-small ◯ a ◯ b √ **c** ◯ d ◯ e

(e) inspect then if o = looks-big then ship-big; if o = looks-small then ship-small; otherwise make-small then ship-small. ◯ a √ **b** ◯ c ◯ d ◯ e

(f) Which policy tree is never helpful? Indicate its associated α vector. ◯ a ◯ b ◯ c √ **d** ◯ e