Northeastern University
CS5100 – Foundations of Artificial Intelligence
Fall 2022 Semester, Richard Hoshino

Problem Set #4

This Problem Set is due at 6PM on **Wednesday November 2**.

This Problem Set will be marked out of 40, and is worth 8% of the final course grade. There are five problems, each worth eight marks.

Problems #1 and #2 and #3 are to be completed individually, while Problems #4 and #5 are to be completed in your assigned small groups. For the two group problems, you are strongly encouraged, though not required, to have a different lead author for each of parts (a), (b), and (c).

Please type (or neatly handwrite) your solutions on standard 8.5×11 paper, with your name(s) at the top of each solution. Ensure that you submit five *separate* PDF files on Canvas, one for each problem. Make sure you label your Problem Set #4 submissions appropriately - e.g. richard4-1.pdf, richard4-2.pdf, richard4-3.pdf, richard4-4.pdf, richard4-5.pdf.

Given that the last two problems are done in a group, your final two PDF files will be identical to some of your classmates. (For example, richard4-4.pdf might be identical to yvonne4-4.pdf and bethany4-4.pdf). This is completely fine, and enables you to have a record of all of your submitted work in this course.

While a solution must be absolutely perfect to receive full marks, I will be generous in awarding partial marks for incomplete solutions that demonstrate progress.

So that there is no ambiguity, there are two non-negotiable rules. A violation of either rule constitutes plagiarism and will result in you receiving an F for this course.

(a) If you meet with a classmate to discuss one of the three Individual Problems, the articulation of your thought process (i.e., what you submit to me), must be an individual activity, done in your own words, away from others. Please remember that the solution-writing process is where so much of your learning will occur in this course: much more than anything we do in class, and even more than the time you spend on solving the problems. Do not be surprised if it takes you 3 to 5 times as long to write up a solution than it takes you to actually solve the problem. (For me, as an academic researcher writing formal proofs for publications, my ratio is significantly higher!)

(b) This Problem Set has been designed to be challenging, because struggling through problems is how we learn best. Your educational experience is cheapened by going online and finding the solution to a problem; even using the Internet to look for a "small hint" is unacceptable. In return, I will be readily available during our optional problem-solving workshops on Monday morning, and upon request, I will post hints to any questions you have on our class Canvas Page.

Problem #1 – INDIVIDUAL

In a 2-player game (with players Rose and Colin), each player chooses either a red card or a black card. The payoff scenarios are as follows:

If Rose chooses red and Colin chooses red, then Rose wins \$$p$ from Colin.

If Rose chooses red and Colin chooses black, then Rose loses \$$q$ to Colin.

If Rose chooses black and Colin chooses red, then Rose loses \$$r$ to Colin.

If Rose chooses black and Colin chooses black, then Rose wins \$$s$ from Colin.

Given this context, answer the following questions.

(a) Suppose that $p = 2$, $q = 1$, $r = 5$, and $s = 2$. Let x be the probability that Rose selects a red card, and let y be the probability that Colin selects a red card. Determine Rose's expected payoff, which will be a formula in terms of x and y.

(b) Compute the Nash Equilibrium of this game, i.e., the "optimal" values of x and y for both Rose and Colin. Clearly explain why your pair (x, y) is a Nash Equilibrium, i.e., neither player can improve their payoff by deviating from their strategy. Finally, determine the expected per-round payoff of this game for each player.

(c) Suppose that p, q, r, s are positive integers such that the Nash Equilibrium of the game leads to an expected per-round payoff is \$0 for each player, i.e., the game is completely fair. Prove that $ps - qr = 0$.

For those of you who have taken a Linear Algebra course, you will recognize that the condition $ps - qr = 0$ is equivalent to the condition that the determinant of the payoff matrix is 0.

Problem #2 – INDIVIDUAL

In this problem, assume that we are given a Markov Decision Process (MDP) with a finite number of states and actions.

(a) Consider an MDP with no terminal states, which means that an agent makes an infinite number of moves, collecting a finite reward $R(s, a, s')$ after each move. If the discount rate γ satisfies $0 < \gamma < 1$, explain why the agent's total utility (i.e., sum of rewards) cannot be infinite.

Hint: do you remember a formula for the infinite geometric series $1 + \gamma + \gamma^2 + \gamma^3 + \ldots$?

(b) Suppose that the values in Value Iteration have just converged, i.e., $V_{k+1}(s) = V_k(s)$ for all states s. Must it be true that the policy has converged as well? If so, explain why we must have $\pi_{k+1}(s) = \pi_k(s)$ for all states s. If not, provide a counterexample.

Suppose that the policy in Policy Iteration has just converged, i.e., $\pi_{k+1}(s) = \pi_k(s)$ for all states s. Must it be true that the values have converged as well? If so, explain why we must have $V_{k+1}(s) = V_k(s)$ for all states s. If not, provide a counterexample.

(c) Consider two MDPs that have the same set of states S, same set of actions A, same transition function T, same reward function R, and only differ in the discount factor γ.

We run Value Iteration on both MDPs until the values have converged. From this process we determine the optimal values $V^*(s)$ and the optimal policy $\pi(s)$ for each MDP.

Suppose the optimal policy for the first MDP is $\pi_1(s)$ and the optimal policy for the second MDP is $\pi_2(s)$.

Must these policies be identical? If so, explain why $\pi_1(s) = \pi_2(s)$ for all states $s \in S$. If not, provide a simple counterexample.

Problem #3 – INDIVIDUAL

I am holding a fair coin, where on each toss, each of the options $\{\text{Heads}, \text{Tails}\}$ comes up with equal probability.

We play the following game, where there is some fixed positive integer n that is our *target score*.

You start with 0 points. On each turn, I flip the coin.

If I toss Heads, you get +2 points. And if I toss Tails, you get +1 point.

If you hit exactly n points, you win the game and your payoff is 1. However, if your total score exceeds n, then you lose the game and your payoff is 0.

You are welcome to stop the game before hitting n points, but if you do then your payoff is 0.

For example, suppose the target score is $n = 5$. Say I flip Tails then Heads then Tails then Tails. The total score is $1 + 2 + 1 + 1 = 5$ and so you win the game.

(a) For each state s, let $V_k(s)$ be the value of the state at step k.

Using the Bellman Equation, clearly explain why $V_{k+1}(s) = \max\left\{V_k(s), \frac{1}{2}V_k(s+1) + \frac{1}{2}V_k(s+2)\right\}$ for all non-negative integers k and s.

(b) Let $n = 5$. By creating a table of values for $V_k(s)$, determine the value of $V^*(0)$, the expected utility for a rational agent playing this game with a target score of 5, starting with 0 points.

Briefly explain why $V^*(0)$ is also equal to the *probability* that a rational agent wins this game, following the optimal policy of PLAYING if the score is less than n and STOPPING if the score is n or more.

(c) Suppose the target score is $n = 1000000000$.

Determine the value of $V^*(0)$. Clearly justify your answer.

For this entire problem, please do it without Python. If you must use Python, then please only use it to verify your answers.

Problem #4 – GROUP

The Colonel Blotto Problem is a competitive zero-sum game on how to best allocate resources.

Say our players are Billy and Sommer, where Billy has 9 total soldiers and Sommer has 7 total soldiers. Each player decides how to distribute their soldiers among two locations, X and Y.

For each site, whoever sends more soldiers wins that site, and captures all of the enemy soldiers. The payoff is one point for each captured soldier and one point for winning that site. If both players send the same number of soldiers, then the payoff is 0 to both players.

For example, if Billy's move is $(X, Y) = (5, 4)$ and Sommer's move is $(X, Y) = (2, 5)$, then Billy's payoff is $(2 + 1) - (4 + 1) = -2$, since he gains three points for his victory at X and loses five points for his defeat at Y. Similarly, Sommer's payoff is $-(2 + 1) + (4 + 1) = 2$.

The goal is for each player to formula a strategy that maximizes their expected payoff. The solution corresponds to the game's *Nash Equilibrium.*

(a) Explain how the Colonel Blotto Problem is applicable in three *different* real-life domains, with one paragraph on each application. There are plenty of real-life applications to choose from, including military defense, advertising, political elections, and professional sports.

(b) In parts (b) and (c), you will solve a smaller version of the Colonel Blotto game, where Billy has four soldiers and Sommer has three soldiers.

Let (a, b, c, d, e) represent Billy's strategy, where a, b, c, d, e respectively represent the probability that Billy sends 0, 1, 2, 3, 4 soldiers to location X and the rest to location Y.

Let (p, q, r, s) denote Sommer's strategy, where p, q, r, s respectively represent the probability that Sommer sends 0, 1, 2, 3 soldiers to location X and the rest to location Y.

Suppose Billy has learned that Sommer's strategy is $(p, q, r, s) = (0, \frac{1}{2}, \frac{1}{2}, 0)$, i.e., in each round of the game, Sommer always places two soldiers in one location and one soldier in the other location.

Given this information, determine Billy's optimal strategy (a, b, c, d, e) that maximizes his expected payoff.

(c) Determine this game's Nash Equilibrium, i.e., the optimal strategies (a, b, c, d, e) and (p, q, r, s) that both players should employ when Billy has 4 soldiers and Sommer has 3 soldiers.

(d) (**OPTIONAL BONUS**) Determine the Nash Equilibrium of the game when Billy has 9 soldiers and Sommer has 7 soldiers.

Problem #5 – GROUP

For this problem all you need to do is submit a single Python (.py) file that answers the questions below. Make sure your program is well-documented so that your TA can follow through your code.

Consider a 4×3 grid. The start state is $[0, 0]$. There are two terminal states: $[3, 1]$ with a reward of -1, and $[3, 2]$ with a reward of $+1$. There does not exist a state at $[1, 1]$.

You will use Markov Decision Processes (MDP) and Reinforcement Learning (RL) to determine the optimal way to navigate this grid, with a discount factor of $\gamma = 0.9$.

In this world, actions do not always go as planned: 80% of the time the agent takes the intended action, but 10% of the time the agent moves ninety degrees clockwise from the intended action, and 10% of the time the agent moves ninety degrees counterclockwise from the intended action. Furthermore, if there is a wall in the direction of where the agent would have gone, the agent stays put in that cell.

For example, if $s = [0, 0]$ and $a =$ WEST, then $T(s, a, [0, 1]) = 0.1$ and $T(s, a, [0, 0]) = 0.8 + 0.1 = 0.9$.

If $s = [0, 0]$ and $a =$ NORTH, then $T(s, a, [0, 1]) = 0.8$, $T(s, a, [1, 0]) = 0.1$, and $T(s, a, [0, 0]) = 0.1$.

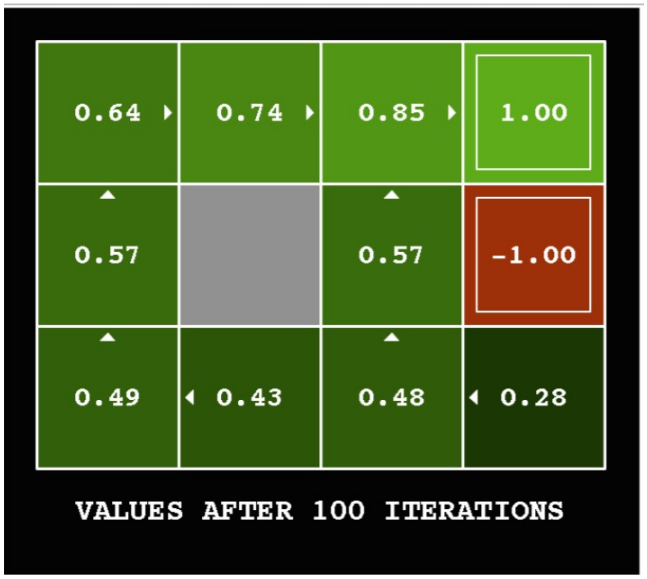
(a) Create an MDP with this set of 12 states:

$$S = [[0, 0], [1, 0], [2, 0], [3, 0], [0, 1], [2, 1], [3, 1], [0, 2], [1, 2], [2, 2], [3, 2], \text{“EXIT”}],$$

and this set of 4 actions: $A =$ [“NORTH”, “EAST”, “WEST”, “SOUTH”].

For each action $a \in A$, we have $R([3, 1], a, \text{“EXIT”}) = -1$ and $R([3, 2], a, \text{“EXIT”}) = +1$, with all other reward values being 0.

For each $s, s' \in S$ and $a \in A$, determine the correct transition probability $T(s, a, s')$ so that we get the following MDP values and policies after 100 iterations, assuming a discount rate of $\gamma = 0.9$.



(b) Temporal Difference learning (**TD learning**) is a powerful model-free RL technique that enables us to estimate the set of values $V^\pi(s)$ for any policy π.

At each timestep, an agent takes the action $a = \pi(s)$ from a state s, transitions to state s' with probability $T(s, a, s')$, and receives a reward of $R(s, a, s')$.

Use a random number generator (e.g. random.random()) to determine your samples. For example, if $s = [0, 0]$ and $a =$ NORTH, then 80% of the time the agent will go to $s' = [0, 1]$, 10% of the time the agent will go to $s' = [1, 0]$, and 10% of the time the agent will stay put at $s' = [0, 0]$.

Using the optimal policy π found in part (a), with a learning rate of $\alpha = 0.02$, determine the set of values $V^\pi(s)$ after $k = 5000$ samples.

Your results should be quite similar to the results you found in part (a). For example, in my Python program, my values were

0.640	0.744	0.855	0.998
0.566		0.575	-0.998
0.485	0.431	0.477	0.254

(c) TD learning enables us to compute the value of all states under the policy they follow, but TD learning can't determine an optimal policy. This motivates **Q-Learning**, a powerful model-free RL technique that enables us to directly learn the q-values of each state-action pair (s, a), without the need to know state values, transition functions, or reward functions.

By computing the q-values for each pair (s, a), we can immediately determine the best action for each state s, which allows us to uncover the optimal policy π. Notice that Q-Learning can learn the optimal policy directly from random sub-optimal actions. This is called *off-policy* learning, unlike TD learning which is called *on-policy* learning.

With a learning rate of $\alpha = 0.02$, determine the set of values $Q(s, a)$ after $k = 5000$ samples.

Your optimal q-value for each state s should be quite similar to the results you found in part (a). For example, in my Python program, my values (and optimal actions marked in bold) were

Cell	North	East	West	South
[0,0]	**0.485**	0.405	0.442	0.434
[1,0]	0.395	0.405	**0.430**	0.393
[2,0]	**0.454**	0.299	0.402	0.394
[3,0]	-0.629	0.181	**0.278**	0.271
[0,1]	**0.549**	0.501	0.497	0.452
[2,1]	**0.577**	-0.620	0.508	0.249
[3,1]	-0.998	-0.998	-0.998	-0.998
[0,2]	0.572	**0.630**	0.556	0.523
[1,2]	0.662	**0.756**	0.582	0.668
[2,2]	0.779	**0.874**	0.673	0.546
[3,2]	1.000	1.000	1.000	1.000