Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC3675 - Aprendizaje Reforzado
21 de agosto, 2024

Tarea 2: MDP & MC

Javier Campos A. & Pedro Palma V.

El presente informe condensa, en orden, nuestras respuestas a las preguntas del enunciado. El código que acompaña a los desarrollos está disponible en nuestro Github.

a)

Debemos iniciar utilizando la definición de $v_\pi(s)$

$$v_\pi(s) = \mathbb{E}[G_t \mid S_t = s] \tag{1}$$

Utilizando la ley de probabilidades totales en (1), se obtiene que

$$v_\pi(s) = \sum_{a \in A} \mathbb{E}[G_t \mid S_t = s, A_t = a]\mathbb{P}(A_t = a \mid S_t = s)$$

Por la definición de $q_\pi(s, a)$, se puede reemplazar por la esperanza

$$v_\pi(s) = \sum_{a \in A} q_\pi(s, a)\mathbb{P}(A_t = a \mid S_t = s)$$

Finalmente, $\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s)$ lo que resulta en

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) q_\pi(s, a)$$

b)

Iniciamos utilizando la definición de $q_\pi(s, a)$

$$q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \tag{2}$$

Utilizando la notación recursiva de los retornos en (2) resulta

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$$

Por las propiedades de la esperanza, podemos separar la expresión en dos esperanzas como sigue:

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] + \gamma\mathbb{E}[G_{t+1} \mid S_t = s, A_t = a] \tag{3}$$

Para continuar la demostración deberemos desarrollar cada una de las esperanzas de (3). Para ello utilizaremos, en primer lugar, la definición para la esperanza de R_{t+1}

$$\mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in R} r\mathbb{P}(R_{t+1} = r \mid S_t = s, A_t = a)$$

Podemos calcular la probabilidad de la expresión anterior como la suma en todo el espacio de estados para S_{t+1}

$$\mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in S} \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

Reordenando y simplificando, se obtiene lo siguiente:

$$\mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in R} \sum_{s' \in S} r p(s', r|s, a) \tag{4}$$

En segundo lugar, se puede utilizar la ley de probabilidades totales en la otra esperanza de (3), resultando en:

$$\mathbb{E}[G_{t+1} \mid S_t = s, A_t = a] = \sum_{s' \in S} \mathbb{E}[G_{t+1} \mid S_{t+1} = s', S_t = s, A_t = a] \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$$

Por la propiedad Markoviana, G_{t+1} no depende de S_t ni de A_t, por lo que

$$\mathbb{E}[G_{t+1} \mid S_t = s, A_t = a] = \sum_{s' \in S} \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$$

Utilizando la definición de $v_\pi(s)$ definida en (1), queda la siguiente expresión

$$\mathbb{E}[G_{t+1} \mid S_t = s, A_t = a] = \sum_{s' \in S} v_\pi(s') \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$$

Ahora, podemos calcular la probabilidad de la expresión anterior como la suma en todo el espacio de recompensas para R_{t+1}

$$\mathbb{E}[G_{t+1} \mid S_t = s, A_t = a] = \sum_{s' \in S} v_\pi(s') \sum_{r \in R} \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

Reordanando y simplificando se obtiene la siguiente expresión

$$\mathbb{E}[G_{t+1} \mid S_t = s, A_t = a] = \sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) v_\pi(s') \tag{5}$$

Finalmente, reemplazamos (4) y (5) en (3)

$$q_\pi(s, a) = \sum_{r \in R} \sum_{s' \in S} r p(s', r|s, a) + \gamma \sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) v_\pi(s')$$

Uniendo las sumatorias y factorizando por $p(s', r|s, a)$, resulta:

$$q_\pi(s, a) = \sum_{r \in R} \sum_{s' \in S} p(s', r|s, a)(r + \gamma v_\pi(s'))$$

c)

El MDP descrito se puede puede observar en la Figura 1

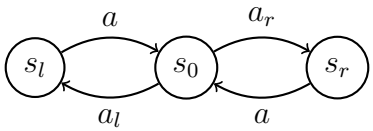

Figura 1: Grafo del MDP determinísico

Al considerar $\gamma = 0$ solo estaremos considerando la recompensa inmediata en nuestro retorno. Para este caso la política óptima sería π_l, puesto que al ejecutar a_l desde s_0 obtendremos una recompensa de 1, lo cual es mayor a la recompensa de 0 que obtendríamos al ejecutar a_r. Luego, para $\gamma = 0.9$ obtenemos que el retorno esperado para las políticas es:

$$\pi_r : \sum_{t=0}^{\infty} 0.9^{2t} \cdot 2 = 10.5263$$
$$\pi_l : \sum_{t=0}^{\infty} 0.9^{2t-1} = 5.8479$$

Por consiguiente, la política óptima es la que posee mayor retorno esperado la cual es π_r.

Finalmente, al considerar un reotrno esperado con $\gamma = 0.5$ obtenemos los siguientes valores para las políticas:

$$\pi_r : \sum_{t=0}^{\infty} 0.5^{2t} \cdot 2 = 2.6667$$
$$\pi_l : \sum_{t=0}^{\infty} 0.5^{2t-1} = 2.6667$$

En este caso ambas son políticas óptimas puesto que tienen un mismo retorno esperado.

d)

A continuación (Tablas 1-3) se presentan los resultados de evaluar una política uniformemente aleatoria con *iterative policy evaluation* para cada MDP.

size	$V(s_0)$	time [ms]
3	−7.999	2.52
4	−17.999	8.65
5	−37.333	26.92
6	−60.230	65.02
7	−93.496	129.44
8	−131.193	240.31
9	−180.001	390.08
10	−233.879	662.84

Tabla 1: Resultados GridProblem $\gamma = 1$

size	$V(s_0)$	time [ms]
3	5.380	242.87
4	2.477	768.39
5	1.291	1835.74
6	0.729	3740.77
7	0.436	6615.53
8	0.273	10916.89
9	0.177	17013.17
10	0.117	25747.93

Tabla 2: Resultados CookieProblem $\gamma = 0.99$

p	$V(s_0)$	time [ms]
0.25	0.067	121.34
0.4	0.283	183.01
0.55	0.611	242.33

Tabla 3: Resultados GamblerProblem $\gamma = 1$

e)

Los problemas que demoran más tiempo son los que tienen más estados sobre los cuales iterar, el hecho de tener calcular para más estados hace más lenta la convergencia del algoritmo. También influye el valor de descuento (γ), porque regula cuanto "pesa.[el] valor de estados futuros en la estimación de cada estado. Un γ menor hace más fácil converger, porque el valor estimado para el estado s es menos sensible a las variaciones en los estados s' posibles según la dinámica del sistema. El problema que más tiempo toma con Iterative Policy Evaluation es CookieProblem, con size = 10, que demora más de 25 segundos en converger. Esto comprueba que un problema con mayor número de estados y alto *gamma* toma mas tiempo en ser resuelto, pero, ¿por qué CookieProblem, para cada tamaño, toma 2 ´rdenes de magnitud más de tiempo que GridProblem? Hipotetizamos que se debe a la naturaleza dinámica de CookieProblem, en donde cada vez que el agente encuentra la galleta, esta cambia de lugar a una nueva casilla aleatoria. En cambio, las recompensas de GridProblem estan fijas en la grilla. Es natural que la estimación de valor demore más en converger cuando la recompensa de un cierto está asociada a una galleta que cambia de posición.

f)

Al disminuir el valor de la tasa de descuento, es posible observar que el algoritmo *iterative policy evaluation* reduce su tiempo para resolver el problema. Esto se debe a que al tener una tasa de descuento más pequeña, las recompensas de más largo plazo se vuelven menos significativas para la estimación de los $V(s)$. Asimismo, este efecto es más significativo en problemas con una mayor cantidad de estados, como el GridProblem o el CookieProblem con un gran tamaño. Es importante destacar el impacto que existe en este último problema, puesto que al no ser episódico la tasa de descuento es muy importante para hacer que la sumatoria para nuestra estimación de los $V(s)$ converja.

g)

A continuación se muestran los resultados obtenidos al evaluar la política *greedy* con *iterative policy evaluation*, para cada MDP (Tablas 4-6). Para evaluar la optimalidad de la política *greedy*, se compara con los resultados de $V^*(s)$ $\forall s \in \mathcal{S}$ obtenidos del algoritmo *Value Iteration*, que por diseño converge a una política óptima. Es decir, podemos decir que el valor $V(s)$ (calculado por *iterative policy evaluation*) es óptimo si se cumple que $V(s) = V^*(s)$.

size	$V(s_0)$	Óptima?
3	-2.0	si
4	-2.0	si
5	-4.0	si
6	-4.0	si
7	-6.0	si
8	-6.0	si
9	-8.0	si
10	-8.0	si

Tabla 4: Resultados de política *greedy* en GridProblem $\gamma = 1$

size	$V(s_0)$	Óptima?
3	48.759	si
4	35.906	si
5	28.197	si
6	23.063	si
7	19.401	si
8	16.661	si
9	14.534	si
10	12.838	si

Tabla 5: Resultados de política *greedy* en CookieProblem $\gamma = 0.99$

p	$V(s_0)$	Optima?]
0.25	0.25	si
0.4	0.4	si
0.55	0.729	no

Tabla 6: Resultados de política *greedy* en GamblerProblem $\gamma = 1$

h)

A continuación, se reportan los resultados del algoritmo Value Iteration en cada uno de los MDPs (Tablas 7-9).

size	$V(s_0)$	time [ms]
3	-2.0	21.57
4	-2.0	22.52
5	-4.0	22.64
6	-4.0	22.94
7	-6.0	24.59
8	-6.0	23.40
9	-8.0	28.94
10	-8.0	32.09

Tabla 7: Resultados de *Value Iteration* en GridProblem $\gamma = 1$

size	$V(s_0)$	time [ms]
3	48.759	528.49
4	35.906	1020
5	28.197	2610
6	23.063	5350
7	19.401	10600
8	16.661	16060
9	14.534	26030
10	12.838	41170

Tabla 8: Resultados de *Value Iteration* en CookieProblem $\gamma = 0.99$

p	$V(s_0)$	time [ms]
0.25	0.25	55.80
0.4	0.4	76.46
0.55	0.9999	831.78

Tabla 9: Resultados de *Value Iteration* en GamblerProblem $\gamma = 1$

i)

En los gráficos de las figuras 2-4 documentamos el set de políticas óptimas para GamblerProblem variando $p \in \{0.25, 0.4, 0.55\}$.

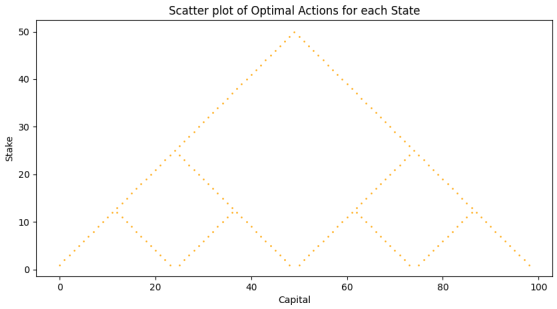

Figura 2: Montos óptimos de apuesta para cada estado en GamblerProblem, $\gamma = 1, p = 0.25$

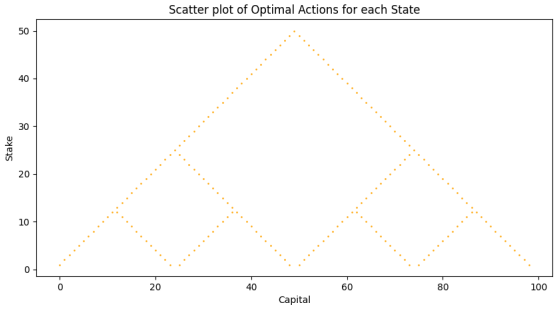

Figura 3: Montos óptimos de apuesta para cada estado en GamblerProblem, $\gamma = 1, p = 0.4$

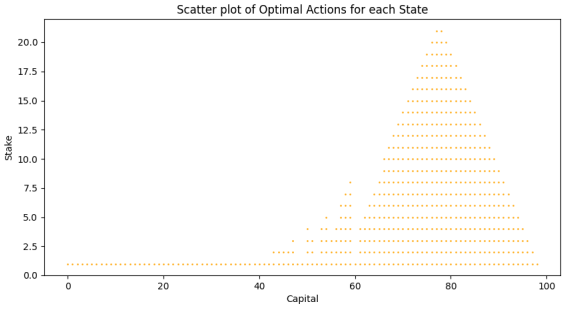

Figura 4: Montos óptimos de apuesta para cada estado en GamblerProblem, $\gamma = 1, p = 0.55$

j)

A continuación, reportamos el desempeño del algoritmo Monte Carlo en los ambientes Black-Jack y Cliff (Figuras 5 y 6).

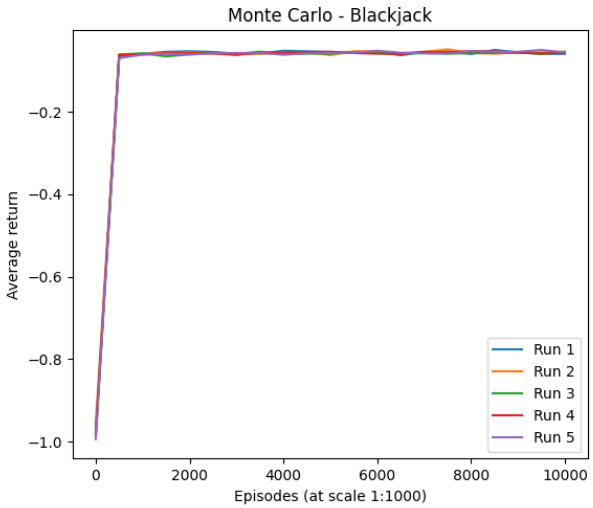

Figura 5: Desempeño (recompensa media) MonteCarlo en ambiente BlackJack, $\gamma = 1$, $\epsilon = 0.01$. 5 runs de 10 millones de episiodios c/u

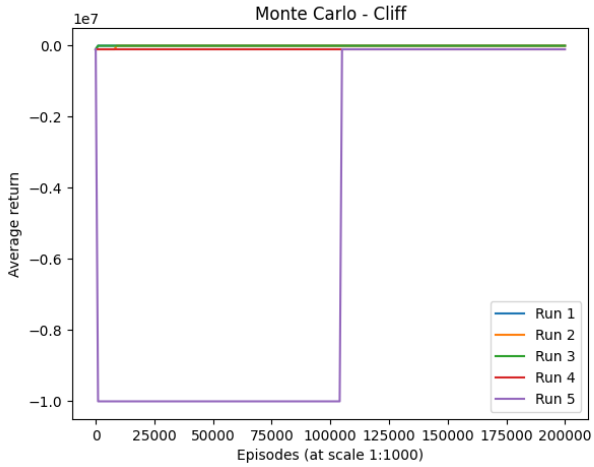

Figura 6: Desempeño (recompensa media) MonteCarlo en ambiente Cliff, $\gamma = 1$, $\epsilon = 0.1$. 5 runs de 200 mil episiodios c/u

k)

Desde el gráfico, notamos que luego de los primeros 500000 episodios, el método MonteCarlo alcanza una recompenza esperada cercana a 0. Es decir que, en promedio, empata con el dealer. Si observamos los q-values usando *print_value= True*, podemos ver que solo se arriesga a ejecutar hit si el valor de la carta del dealer es bajo (< 6) mientras que la suma de sus propias cartas no

sea mayor a 18. En el resto de ocasiones parece aprender a ser conservador y decidir stick.
Notamos que el comportamiento es muy similar en las 5 runs, por lo que podemos decir que Monte-Carlo produce resultados consistentes. Uno de los motivos que más contribuye a esta consistencia es $\epsilon = 0.01$, que representa un porcentaje muy bajo de exploración, seleccionando una política greedy la gran mayoría del tiempo.

l)

Figura 7: Politica Monte Carlo para Cliff

Al inspeccionar la política para los últimos episodios, observamos que el agente despega desde el inicio alejándose del borde y busca aterrizar en el cuadro de meta, evitando siempre la cercanía con el cliff. Desde el gráfico del rendimiento promedio según los episodios, en 5 runs distintas, podemos ver que MonteCarlo parece no conseguir resultados consistentes. Creemos que una posible explicación para esta irregularidad es la baja cantidad de episodios de aprendizaje, en conjunto a un 10 % de exploración aleatoria que terminan causando que la política aprendida no sea del todo eficiente.

m)

Si bien Montecarlo es útil en determinados contextos, no lo consideraríamos un algoritmo estable. Es muy propenso a encontrar politicas sub-óptimas durante su exploración, en donde puede quedarse "pegado", demorando así un tiempo considerable en obtener resultados aceptables.

n)

Al hacer el experimento con cliff de ancho = 12, notamos que el algoritmo toma mucho más tiempo en terminar su ejecución. Observamos también que parece quedarse estancado en un valor bastante negativo de recompensa promedio, lo cual indica que no está aprendiendo a resolver el problema de forma inteligente. Es capaz de encontrar una forma de resolver el problema, pero no parece óptima y probablemente requiera de un mayor número de episodios - y con ello **aún más tiempo** para converger.