Pontificia Universidad Católica de Chile
Departamento de Ciencias de la Computación
IIC3675- Aprendizaje Reforzado

Tarea 3

Integrantes: Matías Orbeta, Sebastián Terrazas

---

a) ¿Por qué Q-Learning se considera un off-policy method mientras que Sarsa se considera un on-policy method?

- **Q-Learning es off-policy** porque en la corrección de TD utiliza siempre la acción de mayor valor estimado ($\max_{a'} Q(s', a')$), independientemente de la acción realmente tomada por la política de comportamiento.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$$

- **SARSA es on-policy** porque su actualización emplea precisamente la acción a' que la política de comportamiento eligió en el siguiente estado, incluyendo los pasos exploratorios (ε-greedy).

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$$

b) Considera el caso en que la selección de acciones es greedy. ¿Son Sarsa y Q-Learning equivalentes en este caso (i.e., eligen las mismas acciones y realizan las mismas actualizaciones en todo momento)? Si la respuesta es sí, demuestra que son equivalentes. Si es no, muestra un caso en que no sean equivalentes

- Para Q,ajo selección **greedy**, la acción tomada en el estado s' es siempre

$$a' = \arg\max_{a'} Q(s', a')\,.$$

- Entonces, el *target* de Q-Learning,

$$r + \gamma \max_{a'} Q(s', a'),$$

coincide exactamente con el de SARSA,

$$r + \gamma Q\big(s',\, a'\big) = r + \gamma Q\big(s',\, \arg\max_{a'} Q(s', a')\big).$$

- Por tanto, SARSA y Q-Learning realizan exactamente la misma actualización paso a paso cuando no hay exploración

c) Compara el rendimiento de Q-learning, Sarsa y 4-step Sarsa en el dominio CliffEnv. Utiliza $\epsilon = 0{,}1$ para explorar, un learning rate de $\alpha = 0{,}1$ y descuento $\gamma = 1{,}0$. Inicializa los Q-values en 0. Corre 100 veces cada experimento y grafica el retorno promedio por episodio para los primeros 500 episodios. En tu gráfico, fuerza a que el eje-Y parta en -200 para que sea más fácil ver la parte más interesante de los resultados. Explica en detalle por qué los resultados que obtuviste están bien sustentados por la teoría. Por ejemplo, digamos que Q-learning le gana a Sarsa y Sarsa le gana a 4-step Sarsa. Explica por qué Q-learning le debería ganar a Sarsa y luego explica por qué Sarsa le debería ganar a 4-step Sarsa.

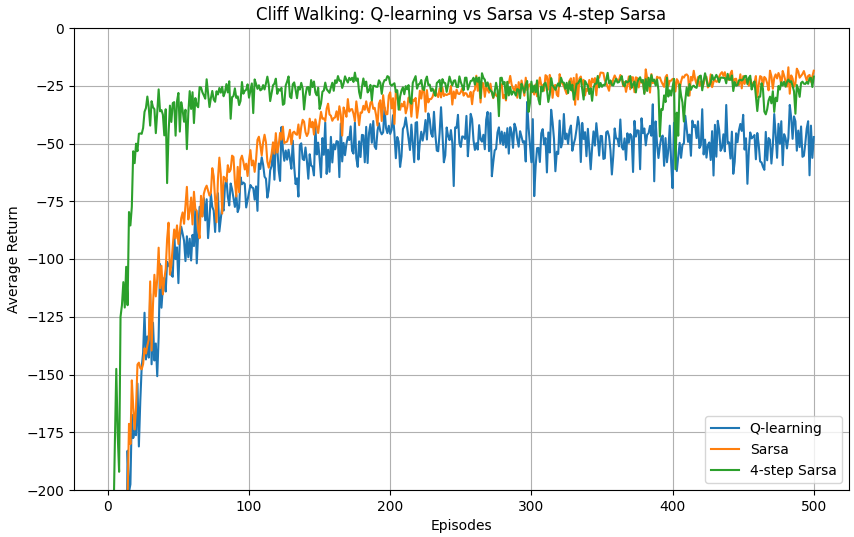

Figura 1: Comparación entre algoritmos *Cliff-Walking*

En los resultados se observa que 4-step Sarsa acelera el aprendizaje inicial en comparación a los otros agentes, pero luego con los episodios, Sarsa normal empieza a tener mejores resultados que este, observándose una tendencia de mejora más que 4-step. Estos dos ïnternalizan.º consideran el riesgo de caer al acantilado. Se ve que ambos tipos de Sarsa le ganan a Q Learning. Esto se puede explicar debido a que Q-Learning usa como target la acción óptima que es el avanzar pegado al cliff y este no considera que con una probabilidad del 10 % explorará y se caerá al acantilado. Pese a esto, Q-Learning actualiza hacia la acción greedy futura, por lo que ignorará esta caída. Debido a esto, Q-Learning tenderá a tener un pero rendimiento en aprendizaje, pero al momento de realizar la explotación de la política greedy, se verá que Q Learning logra los mejores resultados al tener una política greedy que va por el borde del cliff. En cambio, 4-step Sarsa y Sarsa buscarán evitar el borde, obteniéndose un peor resultado de esta forma.
En resumen, en el aprendizaje Sarsa entrega mejores resultados medios luego de los 500 episodios, le sigue 4-step Sarsa y por último se tiene Q-Learning. Pero al explotar la política greedy, se obtendrán los mejores resultados con Q-Learning.

d) Compara el rendimiento de Dyna y RMax en el dominio EscapeRoomEnv. Corre cada método 5 veces por 20 episodios. Usa $\gamma = 1{,}0$. Para Dyna utiliza $\alpha = 0{,}5, \epsilon = 0{,}1$ e inicializa los Q-values en 0. Reporta en una tabla el retorno medio por episodio de RMax y Dyna utilizando 0, 1, 10, 100, 1000 y 10000 pasos de planning. ¿Es verdad que Dyna se vuelve equivalente a RMax cuando el número de pasos de planning es suficientemente grande? Justifica tu respuesta.

Luego de correr los modelos, se obtiene:

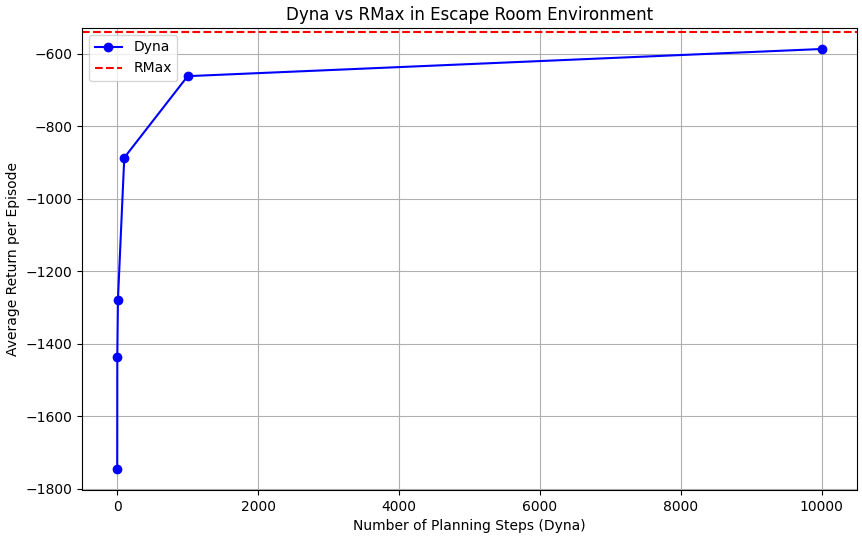

Figura 2: Comparación entre algoritmos en *Escape Room*

De esto se observa que con 20 episodios, Dyna mejora drásticamente el retorno promedio por episodio a medida que aumenta el número de steps de planning, observándose claramente la tendencia al alza (en términos de mejorar). RMax, al ser un modelo optimista, converge de manera más rápido ya que explora todos los estados para lograr.
Con estos resultados y la teoría, se podría sugerir que Dyna RMax con un planning muy grande. Si Dyna realiza suficientes actualizaciones de planificación entre pasos reales, terminará evaluando el modelo de manera completa/exhaustiva el modelo. Cuando el número de steps tiende al infinito (o tan grande para cubrir todas las posibles transiciones conocidas varias veces entre pasos reales), la política resultante y los valores serán iguales (o prácticamente iguales) que el de RMax.
Cabe recalcar que se observa retorno medio por episodio de -600 (para resumir) hacia abajo debido que se consideran los retornos de todos los episodios y esos se dividen por la cantidad de episodios, en este caso, 20 episodios por corrida.

e) Resuelve el dominio *RoomEnv* usando Q-learning, Sarsa, 8-step Sarsa y versiones *multi-goal* de Q-learning y Sarsa. Utiliza $\alpha = 0{,}1$, $\epsilon = 0{,}1$, $\gamma = 0{,}99$ e inicializa los Q-values en 1,0. Corre cada método 100 veces por 500 episodios cada uno. Luego grafica el largo promedio de los episodios de cada algoritmo. ¿Son las versiones *multi-goal* efectivamente mejores que los *baselines* (i.e., las versiones estándar de Q-learning, Sarsa y 8-step Sarsa)? ¿Entre utilizar *multi-goal* Q-learning vs *multi-goal* Sarsa, cuál es mejor? ¿Por qué?

Basado en los resultados mostrados en el gráfico 3, podemos observar:

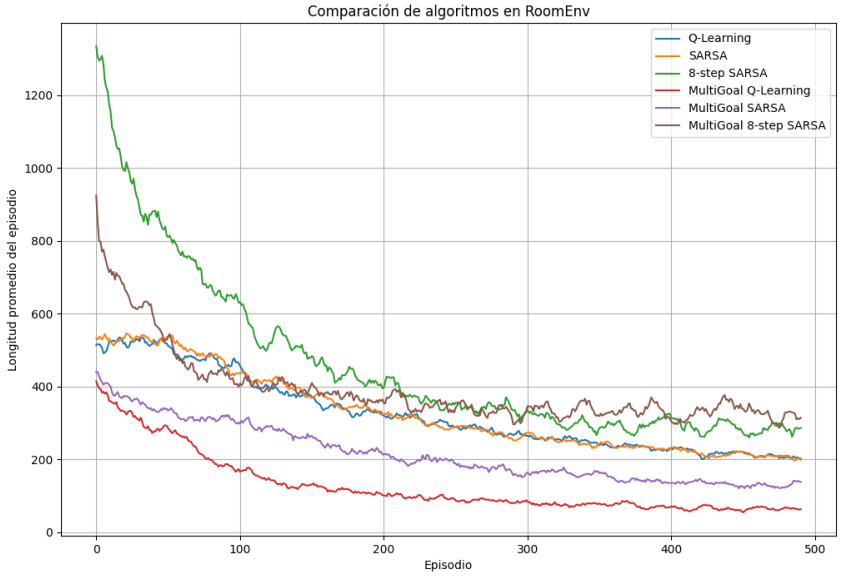

Figura 3: Comparación entre algoritmos y sus variantes *multi-goals*

- Las versiones multi-goal son claramente superiores a sus contrapartes estándar. MultiGoal Q-Learning (rojo) y MultiGoal SARSA (morado) convergen a episodios significativamente más cortos que las versiones estándar de Q-Learning (azul) y SARSA (naranja). Esto demuestra que aprovechar la estructura del problema multi-objetivo, actualizando Q-values para todos los objetivos posibles con cada experiencia, acelera dramáticamente el aprendizaje.

- Entre las versiones multi-goal, Q-Learning supera a SARSA. MultiGoal Q-Learning alcanza episodios más cortos (70) que MultiGoal SARSA (140) al final del entrenamiento. Esto probablemente se debe a que Q-Learning es un algoritmo off-policy que actualiza sus valores Q usando el máximo valor futuro posible, independientemente de la política actual. Esta característica le permite converger más rápidamente hacia la política óptima en este entorno, mientras que SARSA, al ser on-policy, está limitado por la política de exploración que está siguiendo actualmente.

- El algoritmo 8-step SARSA estándar (verde) muestra un comportamiento interesante: comienza con episodios muy largos pero eventualmente mejora. Sin embargo, no logra superar a las versiones multi-goal, confirmando que la actualización simultánea para todos los objetivos es una estrategia superior en este tipo de entornos.

En resumen, para problemas multi-objetivo como RoomEnv, el enfoque multi-goal proporciona una mejora sustancial, y entre ellos, Q-Learning multi-goal ofrece el mejor rendimiento debido a su naturaleza off-policy que permite un aprendizaje más eficiente de la política óptima.

f) Se trabajaron con dominios multi agente. Primero se abarcó el setting de Centralized Cooperative Multi Agent RL, el cual se basa en que un solo agente controla varios agentes al mismo tiempo. Se resolvió el environment CentralizedHunterEnv usando Q-leargning, con los parámetros $\gamma = 0{,}95, \epsilon = 0{,}1, \alpha = 0{,}1$ e inicializando los Q-Values en 1.0. Este se corrió 30 veces por 50000 episodios. Frente a esto, se obtuvo:

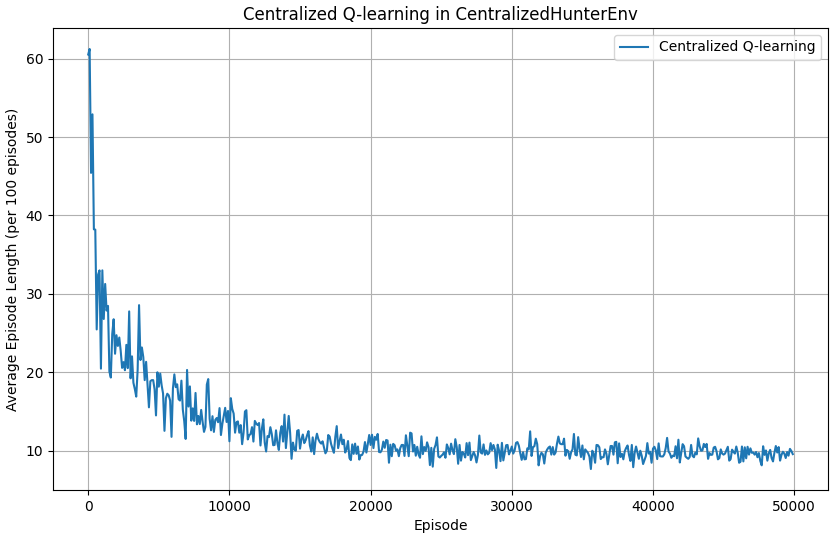

Figura 4: Largo promedio de los episodios (cada 100 episodios) en *CentralizedHunterEnv*

Luego, trabajó con Decentralized Cooperative Multi Agent RL. En este caso ambos agentes comparten la misma función de recompensa, pero seleccionan acciones de manera independiente. Se resolvió el environment HunterEnv usnado dos agentes de Q-learning que controlen, de manera independiente, a cada cazador. Se usaron los mismo parámetros que el caso anterior y se corrió de igual manera 30 veces por 50000 episodios.

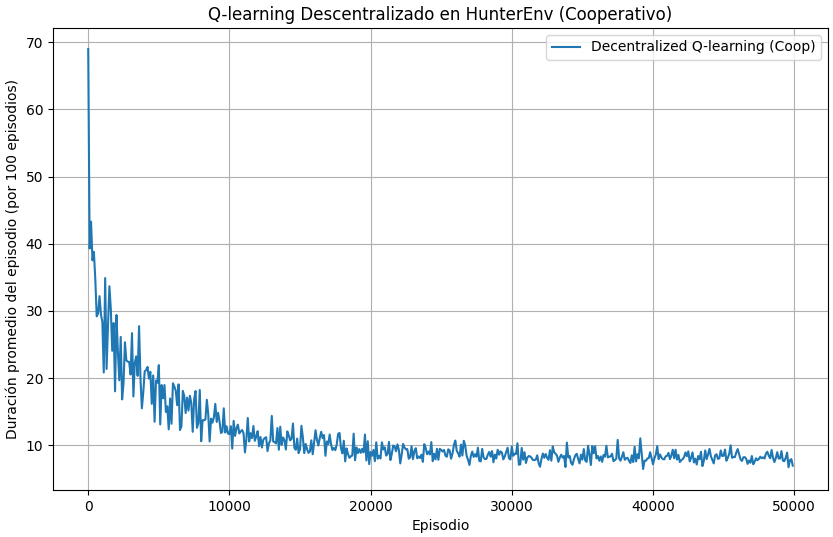

Figura 5: Largo promedio de los episodios (cada 100 episodios) en *HunterEnv*

Por último, se trabajó con Decentralized Competitive Multi Agent RL, donde los agentes compiten y sus funciones de recompensa son opuestas. Se resolvió el dominio HunterAndPreyEnv, donde usaron 3 agentes de Q-Learning independientes, uno para cada cazador y uno para la presa, que también aprende. Se usan los hiperparámetros de los casos anteriores, se corrió 30 veces el experimento por 50000 episodios.

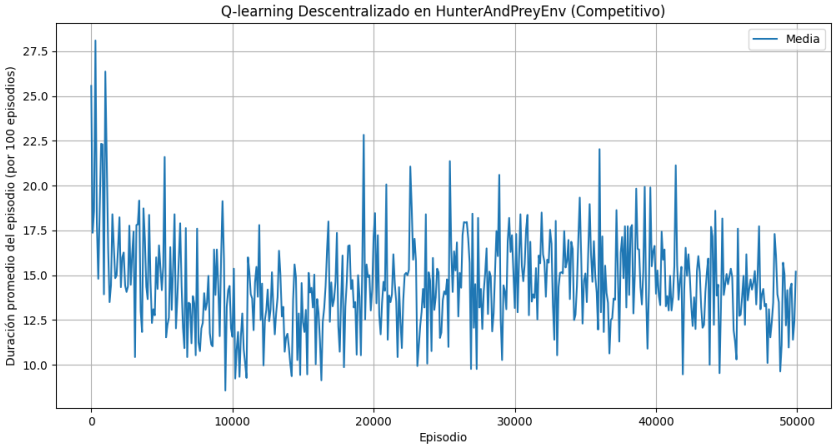

Figura 6: Largo promedio de los episodios (cada 100 episodios) en *HunterAndPreyEnv*

En los resultados se observan tendencias similares entre Centralizado y Descentralizado Cooperativo, donde comienzan con largos promedios de cada episodio altos y luego bajan progresivamente hasta llegar a un promedio cercano a las 10 iteraciones. Si bien se parecen en este comportamiento, Descentralizado logra alcanzar un largo más bajo, lo cual implica que lo resuelve más rápido, llegando a resultados bajos que el caso centralizado, a partir de los episodios 20.000 aproximadamente. Esto se puede explicar a la naturaleza de dos agentes vs un agente, ya que al estar los dos agentes realizando una búsqueda optimizada, con el tiempo ambos lograrán obtener mejor resultados que un solo agente alternando la posición de dos cazadores. En las iteraciones se observó casos donde en centralized un cazador se quedaba detenido y solo se movía el otro, haciendo menos óptima la búsqueda de la presa. Caso contrario es el de dos agentes, donde se observa cómo cada cazador explora por su cuenta, logrando dar más rápidamente con la presa.
El caso de Descentralizado Competitivo es distinto a los otros, ya que parte con una menor duración promedio en comparación con Centralizado y Descentralizado Cooperativo, siendo menos de la mitad de la duración en los primeros episodios, siendo cerca de las 25-30 iteraciones. Pese a esto, se observa que su tendencia es oscilar entre las 10 y 20 iteraciones en la mayoría de los casos. Esto se puede explicar ya que los 3 agentes que participan tienen distintos objetivos. En este dominio, la presa busca evitar ser capturada y los cazadores buscan lograr alcanzarla lo más rápido posible, por lo que existirán veces en las cuales la presa logra escaparse por más iteraciones, y al mismo tiempo, casos donde la presa no logrará escaparse y rápidamente es atrapada. Todas estos tiempo dependen de los aprendizajes que vayan aprendiendo los agentes con los episodios.

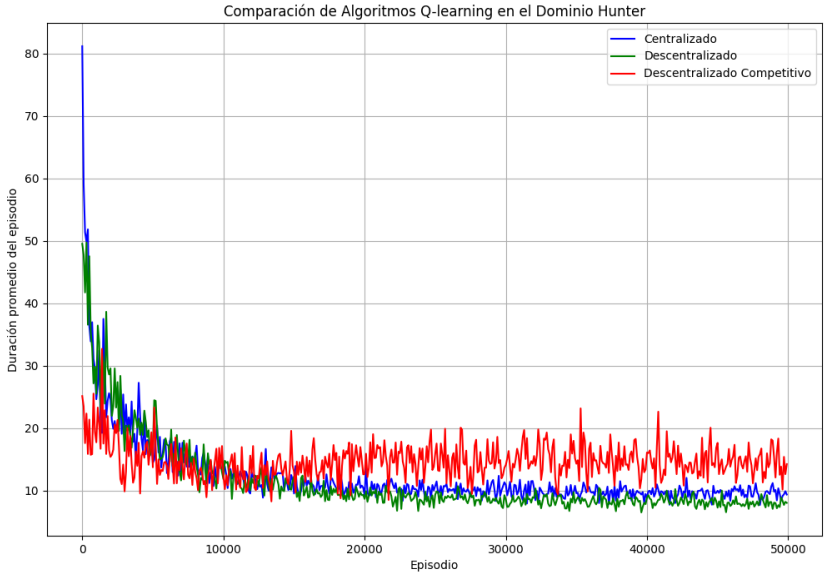

Figura 7: Comparación entre los diferentes settings en dominios multi agente

g) ▪ ¿Existió alguna memoria que sea mejor que otra? ¿A qué se debe esto?

Sí, claramente el **buffer personalizado (Custom Buffer)** superó a las demás memorias, como se puede observar en el gráfico de comparación de la Figura 12. Esta memoria permite al agente alcanzar episodios muy cortos (cercanos a 0-50 pasos) rápidamente, mientras que:

- La memoria sin buffer (No Memory) fracasa completamente con episodios de 5000 pasos
- Las memorias K-order y Binary logran un rendimiento intermedio (∼200 pasos)

Esta superioridad se debe a que:

a) **Control sobre la información:** El buffer personalizado permite al agente decidir **cuándo** guardar información, almacenando solo observaciones realmente relevantes (como el estado de la puerta).

b) **Eficiencia representativa:** Con un solo espacio de memoria, el agente aprende a guardar específicamente la observación crítica que necesitará más tarde (probablemente cuando visita el botón).

c) **Adaptabilidad:** A diferencia de la memoria K-order que guarda automáticamente las últimas k observaciones o la memoria binaria con solo 1 bit, el buffer personalizado combina flexibilidad con precisión.

La Figura 11 muestra claramente esta ventaja, donde todas las líneas (incluyendo Q-Learning y SARSA) convergen a valores muy bajos, algo que no ocurre con los otros tipos de memoria.

▪ ¿Existió algún algoritmo de RL que sea mejor resolviendo estos problemas? ¿A qué se debe esto?

El algoritmo **16-step SARSA** demostró ser consistentemente superior en todos los escenarios de memoria, como se evidencia en todos los gráficos individuales y especialmente en la Figura 8 donde:

- 16-step SARSA (verde) alcanza rápidamente el límite de episodios (lo que indica que no puede resolver el problema sin memoria)
- Q-Learning (azul) y SARSA (naranja) muestran un deterioro más gradual

La superioridad de 16-step SARSA se debe a:

a) **Aprendizaje con horizonte extendido:** Al considerar 16 pasos en el futuro, el algoritmo conecta mejor causas y efectos distantes, crucial en problemas con recompensa retrasada como este.

b) **Mejor manejo de la incertidumbre:** En entornos parcialmente observables, el n-step SARSA promedia sobre múltiples transiciones, reduciendo el impacto de observaciones incompletas.

c) **Actualización más estable:** Como se observa en las Figuras 10 y 9, 16-step SARSA muestra menos fluctuaciones y deterioro a largo plazo.

d) **Mejor generalización:** La técnica n-step permite una propagación más eficiente de valores a través del espacio de estados, especialmente importante con memorias limitadas.

En conclusión, la combinación de buffer personalizado con 16-step SARSA proporciona la mejor solución para este entorno parcialmente observable, demostrando la importancia tanto del mecanismo de memoria como del algoritmo de aprendizaje para abordar estos desafiantes problemas.

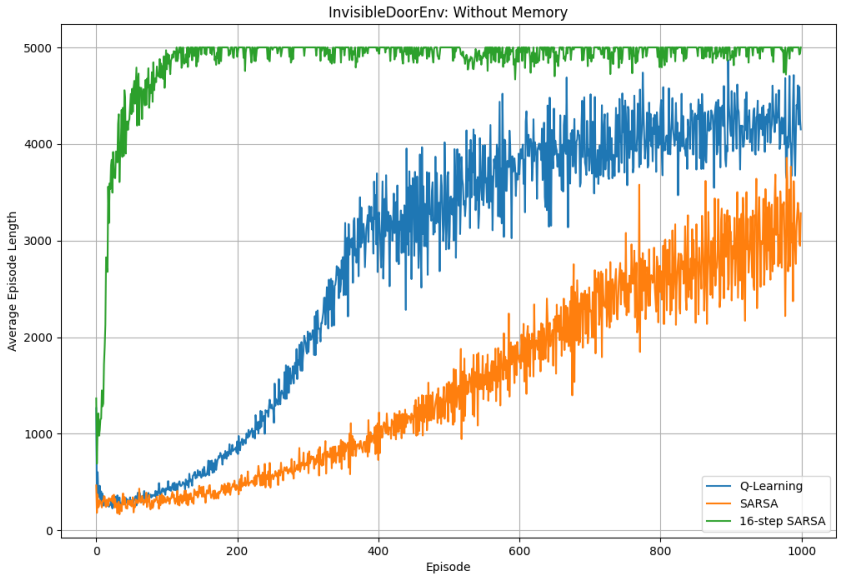

Figura 8: Largo promedio de los episodios para algoritmos sin memoria en *InvisibleDoorEnv*

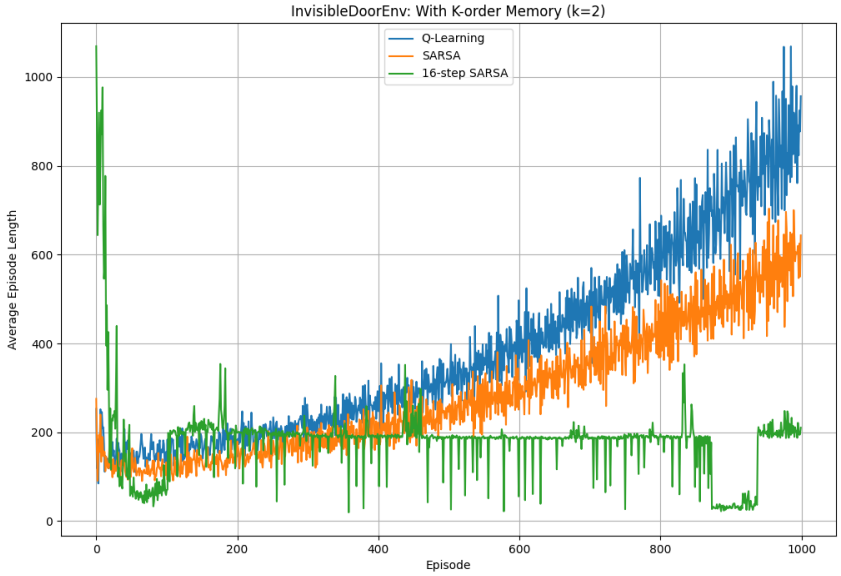

Figura 9: Largo promedio de los episodios utilizando memoria K-order (k=2) en *InvisibleDoorEnv*

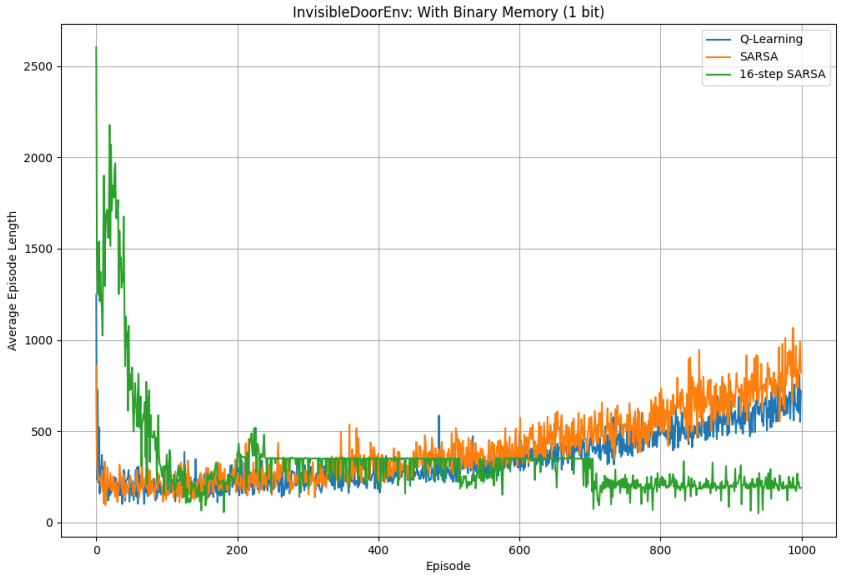

Figura 10: Largo promedio de los episodios utilizando memoria binaria (1 bit) en *InvisibleDoorEnv*

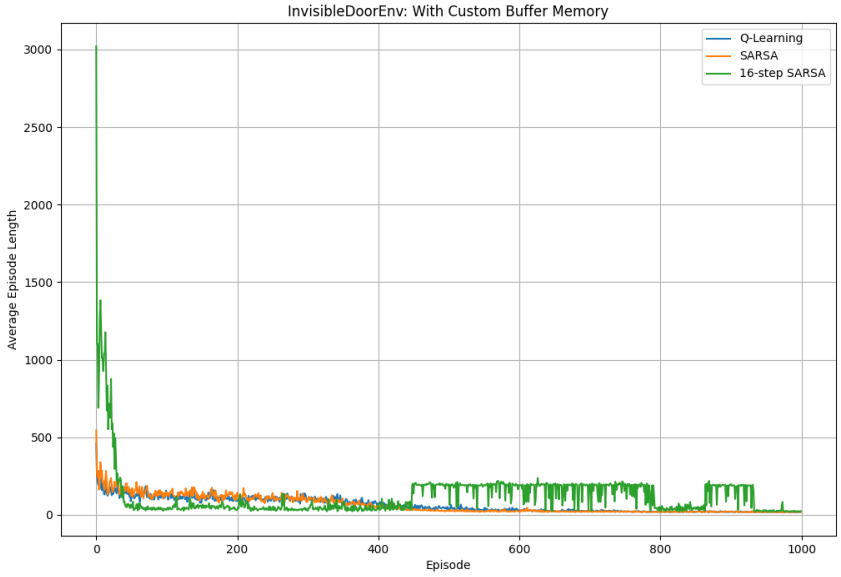

Figura 11: Largo promedio de los episodios utilizando buffer personalizado en *InvisibleDoorEnv*

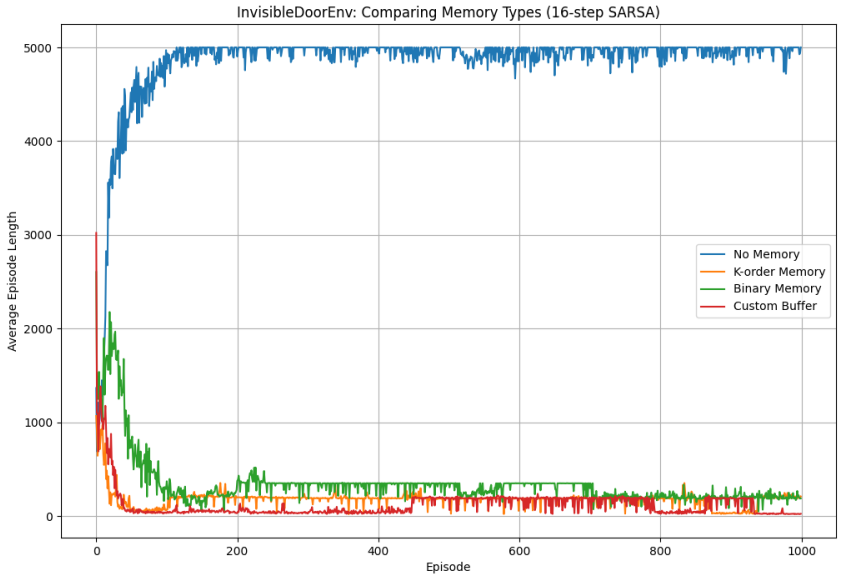

Figura 12: Comparación de diferentes tipos de memoria utilizando 16-step SARSA en *InvisibleDoorEnv*