

# IIC3675: Tarea 3

Bruno Cerdá Mardini

a)

**Q-learning** es *off-policy* ya que en su función de actualización, el valor futuro de la state-action value function ( $Q(S', A')$ ) se elige de manera *greedy*. En específico, se elige la acción que maximiza dicho valor, como se ve en la fórmula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

**SARSA** es *on-policy* ya que, a diferencia de Q-learning, el valor futuro de la state-action value function ( $Q(S', A')$ ) se determina eligiendo la siguiente acción según la política que se está optimizando actualmente:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

b)

Si ambos algoritmos empiezan a elegir acciones de manera greedy, entonces se vuelven muy similares, pero de todas formas **no van a ser equivalentes**.

Esto debido al orden en que se toman las acciones. En **SARSA**, se elige una acción ( $A'$ ) para el siguiente estado ( $S'$ ), luego se hace la actualización, y después esta misma acción ( $A'$ ) se utiliza en el siguiente paso. En cambio, **Q-learning** NO elige una acción ( $A'$ ) y se hace la actualización basado en la acción que simplemente maximice el valor de Q ( $\max_a Q(S', a)$ ), de esta manera Q-learning no se compromete con ninguna acción, luego en el siguiente paso Q-learning es capaz de elegir una acción pero con la tabla Q ya actualizada.

Un ejemplo claro en donde ambos algoritmos podrían tomar decisiones distintas es en caso de empate de valores, debido a que la política greedy no podría elegir un claro ganador, es una situación en donde ambos algoritmos podrían diverger en resultados y la toma de acciones, el ejemplo es el siguiente:

Supongamos que un agente se encuentra en el estado  $S$ , toma la acción  $A$  y debido a esto ahora se encuentra en el estado  $S'$ . Supongamos también que en la tabla Q, el valor para el estado  $S'$  tiene un empate:

- $Q(S', A_1) = -1$

- $Q(S', A_2) = -1$

Debido a que hay un empate, la política greedy no puede elegir una acción que sea claramente mejor, por lo que lo tiene que hacer de manera uniforme. **SARSA** va a actualizar  $Q(S, A)$ , para esto va a elegir la siguiente acción  $A'$ , pero como hay un empate, supongamos que va a terminar eligiendo  $A_1$ , por lo que en el siguiente paso va a estar obligado a ejecutar  $A_1$ . **Q-Learning** va a actualizar  $Q(S, A)$  usando la acción que maximice el valor de retorno, pero como hay un empate, supongamos que va a tomar la acción  $A_2$  como la acción que maximice el valor de  $Q(S', a)$  (**Pero no la guarda para el siguiente paso**). La actualización para  $Q(S, A)$  de ambos algoritmos va a ser la misma, pero **eligieron acciones distintas**, por lo que no son equivalentes.

c)

En la **Figura 1** podemos observar que SARSA y 4-step SARSA son mejores que Q-learning. En específico, en los primeros episodios, 4-step SARSA obtiene retornos mejores que SARSA y Q-learning, pero después SARSA converge al mismo valor que 4-step SARSA en términos de retorno promedio para los últimos episodios, mientras que Q-learning se mantiene claramente más abajo con peores valores. La principal razón por la cual obtenemos estos resultados es debido a que SARSA y 4-step SARSA son **on-policy**, y Q-learning es **off-policy**.

**Q-learning** va a explorar de manera  *$\epsilon$ -greedy*, pero va a actualizar sus valores  $Q(S, A)$  de manera *greedy*. Debido a esto, el algoritmo va a aprender a tomar el camino más corto, pero al mismo tiempo el más riesgoso, ya que con probabilidad  $\epsilon$  se va a caer al *cliff*, donde va a obtener una recompensa de -100. A diferencia de Q-learning, **SARSA** y **4-step SARSA** son **on-policy**, por lo que van a explorar de una manera  *$\epsilon$ -greedy*, y también actualizarán sus valores  $Q(S, A)$  en base a esa misma política. Lo anterior implica que, al explorar los estados cercanos al *cliff*, estos se actualizarán con malos valores  $Q(S, A)$ , ya que con probabilidad  $\epsilon$  el agente se va a caer. Es por esto que el agente va a terminar aprendiendo un camino más largo y seguro.

Finalmente, la razón por la cual 4-step SARSA obtiene mejores retornos promedio de manera mucho más rápida que SARSA es debido a que las actualizaciones de  $Q(S, A)$  consideran 4 pasos más adelante, por lo que logra aprender de manera más rápida que es mejor alejarse del *cliff* para evitar la recompensa de -100. Considerando todo lo anterior, el resultado que se puede observar en la figura tiene mucho sentido.

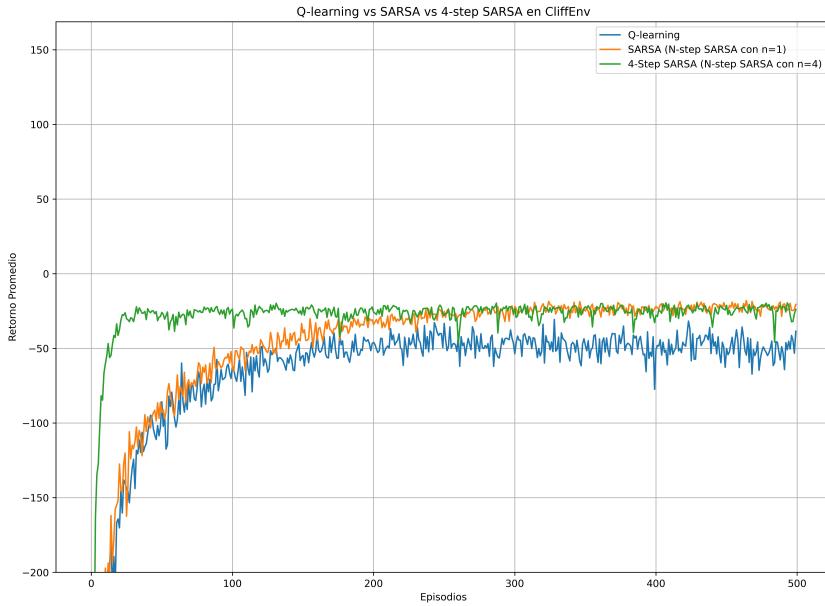


Figure 1: Comparación de retorno promedio entre Q-learning, SARSA y 4-step SARSA en Cliff Walking.

d)

En la Tabla 1 se comparan los rendimientos de Dyna-Q y RMax. En específico, se pueden observar los valores de retorno medio por episodio, donde Dyna-Q tiene múltiples columnas dependiendo de los planning steps y RMax tiene su propia columna.

En el enunciado preguntan si Dyna-Q es equivalente a RMax si se tiene un número lo suficientemente grande. La respuesta es que no, sin importar el número de planning steps, ambos algoritmos van a converger a valores distintos. En específico, RMax va a converger a una política near-optimal, mientras que Dyna-Q va a converger a una política óptima en el límite si explora con  $\epsilon - greedy$ .

Table 1: Dyna y Rmax: Retorno medio por episodio.

| Episodio | n=0     | n=1     | n=10    | n=100   | n=1000  | n=10000 | RMax    |
|----------|---------|---------|---------|---------|---------|---------|---------|
| 1        | -1806.4 | -1910.8 | -2715.4 | -2467.0 | -3145.6 | -2832.4 | -1522.0 |
| 2        | -1599.4 | -2627.2 | -1763.2 | -3126.6 | -3753.2 | -4420.0 | -6922.0 |
| 3        | -2013.4 | -1677.6 | -2497.4 | -2721.8 | -1394.4 | -885.0  | -4843.0 |
| 4        | -1969.0 | -2270.0 | -2053.0 | -1604.8 | -2015.2 | -1195.6 | -804.0  |
| 5        | -2422.0 | -1515.0 | -1298.4 | -1831.2 | -1272.4 | -919.8  | -272.0  |
| 6        | -1057.4 | -1021.4 | -1118.0 | -2468.0 | -156.0  | -288.8  | -232.0  |
| 7        | -1097.2 | -801.8  | -598.6  | -974.2  | -98.0   | -93.8   | -4123.0 |
| 8        | -976.4  | -926.4  | -815.8  | -201.4  | -85.6   | -431.4  | -487.0  |
| 9        | -1106.2 | -752.0  | -848.8  | -189.2  | -86.2   | -81.6   | -285.0  |
| 10       | -996.4  | -797.2  | -1266.0 | -510.0  | -73.6   | -203.8  | -67.0   |
| 11       | -908.6  | -1092.4 | -374.6  | -442.6  | -93.2   | -89.4   | -67.0   |
| 12       | -910.4  | -897.0  | -575.0  | -178.4  | -79.4   | -95.6   | -67.0   |
| 13       | -902.6  | -805.4  | -1764.8 | -127.2  | -91.4   | -169.4  | -67.0   |
| 14       | -827.2  | -1034.6 | -426.4  | -120.4  | -86.8   | -90.4   | -67.0   |
| 15       | -980.6  | -860.2  | -573.6  | -125.6  | -82.8   | -100.4  | -67.0   |
| 16       | -852.2  | -683.6  | -352.6  | -116.2  | -91.6   | -87.4   | -67.0   |
| 17       | -912.0  | -1171.2 | -1174.0 | -109.0  | -85.0   | -96.2   | -67.0   |
| 18       | -1059.6 | -747.0  | -512.8  | -134.2  | -83.6   | -87.8   | -67.0   |
| 19       | -1002.2 | -636.6  | -2148.8 | -92.0   | -82.2   | -83.4   | -67.0   |
| 20       | -1158.0 | -692.2  | -304.8  | -82.6   | -91.6   | -90.8   | -67.0   |

e)

Observando el gráfico, para las baselines podemos afirmar que SARSA y 8-step SARSA no logran aprender de buena manera, ya que sus largos promedios de episodios prácticamente no bajan. Mientras que Q-learning sí logra aprender, evidenciado por su clara disminución del largo promedio de episodio.

Para las versiones multi-goal, la que tiene mejores resultados es Q-learning, ya que de manera clara logra obtener el menor valor para el largo promedio de episodios. 8-step SARSA también logra aprender, obteniendo un rendimiento levemente mejor que el de la versión no multi-goal de Q-learning. Finalmente, SARSA (multi-goal) también muestra aprendizaje, aunque su rendimiento final es inferior al de los otros algoritmos multi-goal.

Considerando lo anterior, sí podemos afirmar que las versiones multi-goal son mejores que sus baselines. Aunque Multi-Goal SARSA es peor que el Q-learning estándar, es un éxito si se compara con su propia baseline (SARSA estándar), la cual no aprende muy bien. De manera similar, Multi-Goal Q-learning y Multi-Goal 8-step SARSA son muy superiores a sus respectivas versiones base.

Al comparar todos los algoritmos, en específico Multi-Goal SARSA vs Multi-Goal Q-learning, este último es claramente el mejor, ya que logra consistentemente el menor largo promedio

de episodio, demostrando ser la estrategia más eficiente para este problema.

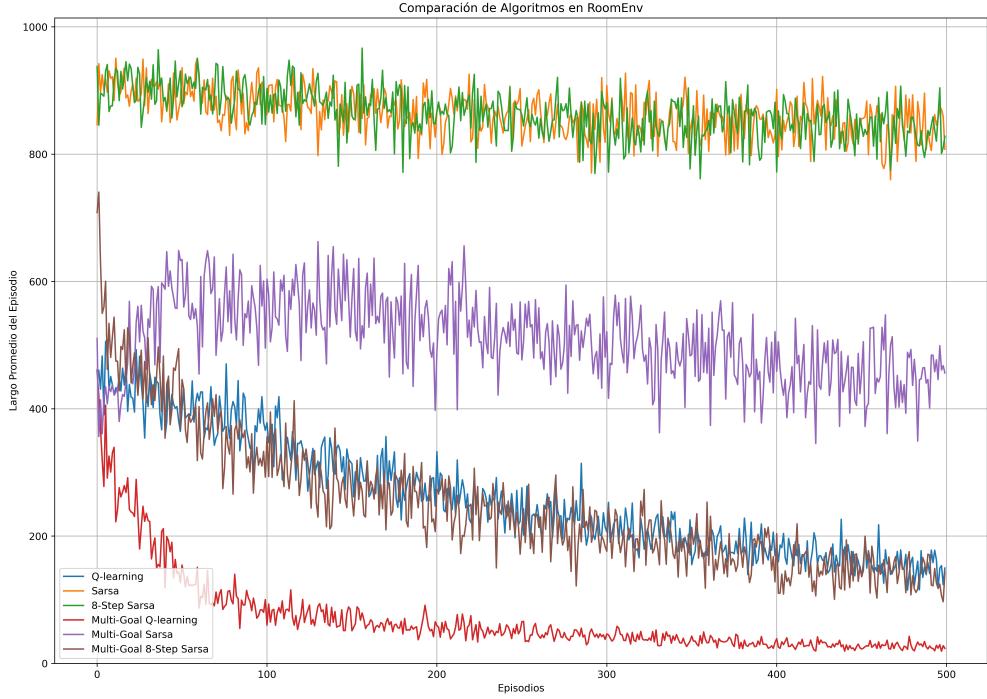


Figure 2: Comparación de Algoritmos en RoomEnv.

f)

En el gráfico, podemos observar que en todos los entornos se logra aprender a capturar a la presa, debido a que todas las curvas logran obtener valores bajos de largo promedio por episodio.

Hay una clara diferencia de comportamiento entre los entornos Centralized y Decentralized Cooperative vs. Decentralized Competitive. Los dos primeros comparten un objetivo, pero lo cumplen de distintas maneras: uno con un solo agente controlando a ambos cazadores y el otro con dos agentes independientes. Es por lo anterior que los entornos cooperativos logran obtener, en promedio, la menor longitud de episodio, ya que aprenden según un mismo objetivo y, lo más importante, la presa no aprende durante este proceso, ya que siempre tiene la misma política.

A diferencia de lo anterior, en el entorno Decentralized Competitive podemos observar que la curva tiene un comportamiento diferente y más inestable. Esto es debido a que la presa va aprendiendo a sobrevivir junto a los cazadores. Es por esta razón que esta curva tiene más ondulaciones que las dos anteriores. En la curva verde, cuando el tiempo promedio de episodio disminuye, quiere decir que los cazadores están aprendiendo a cazar de mejor manera. Pero cuando el tiempo promedio vuelve a aumentar, quiere decir que la presa logró aprender una nueva estrategia para sobrevivir más tiempo.

En resumen, los agentes en los entornos Centralized y Decentralized Cooperative aprenden más rápido que en el Decentralized Competitive, debido a que la presa no va cambiando su política para sobrevivir. Por lo tanto, los agentes aprenden mucho más rápido y de mejor manera cómo cazar a la presa.

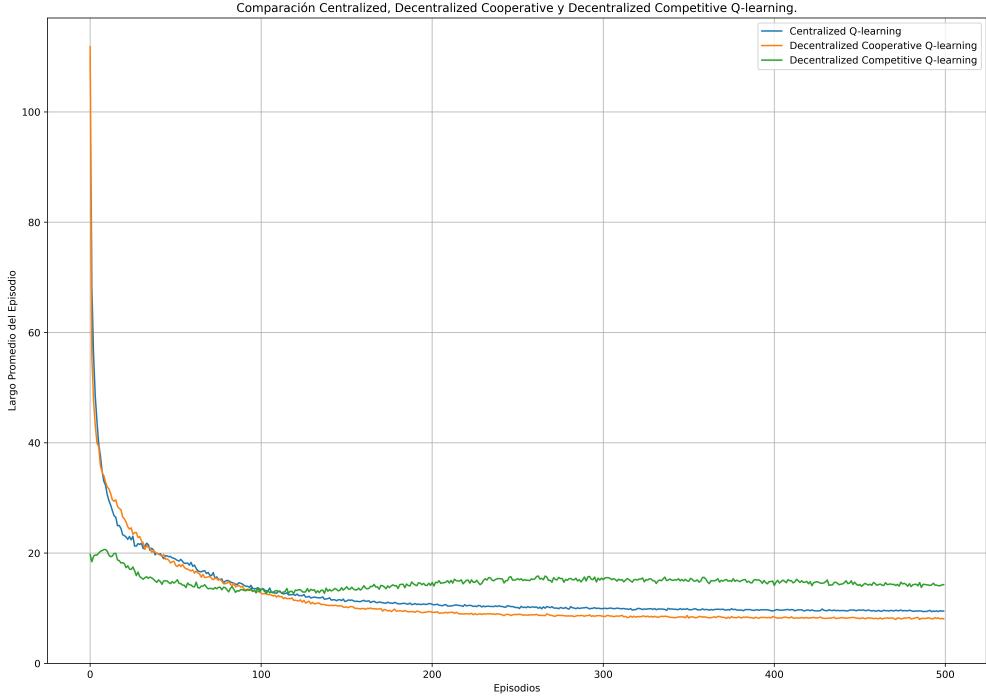


Figure 3: Comparación Centralized, Decentralized Cooperative y Competitive.

g)

La Figura 4 contiene los resultados de SARSA, 16-step SARSA y Q-learning en el InvisibleDoorEnv. Se puede observar que SARSA y 16-step SARSA fallan completamente en el aprendizaje, evidenciado por el constante largo promedio de los episodios, los cuales alcanzan el número máximo de pasos. Q-learning, en los primeros episodios, logra resolver la tarea, pero su rendimiento empeora con el tiempo.

Los comportamientos anteriores tienen una explicación que se relaciona con si el algoritmo es on-policy u off-policy. Dado que SARSA y 16-step SARSA son on-policy, cuando intenten cruzar por la puerta y fallen, su política de exploración se modificará de tal manera que el agente preferirá no acercarse a la puerta, ya que anteriormente no pudo pasar.

A diferencia de lo anterior, Q-learning es off-policy, por lo que, a través de la elección de la acción que tenga el mayor Q-value, a veces el agente logrará cruzar por la puerta y recordará que ese camino es bueno. Sin embargo, como no tiene memoria, no sabe que para lograr eso es necesario apretar el botón, por lo que comenzará a quedarse pegado cerca de la puerta. Es por esta razón que su rendimiento va empeorando a lo largo del tiempo.

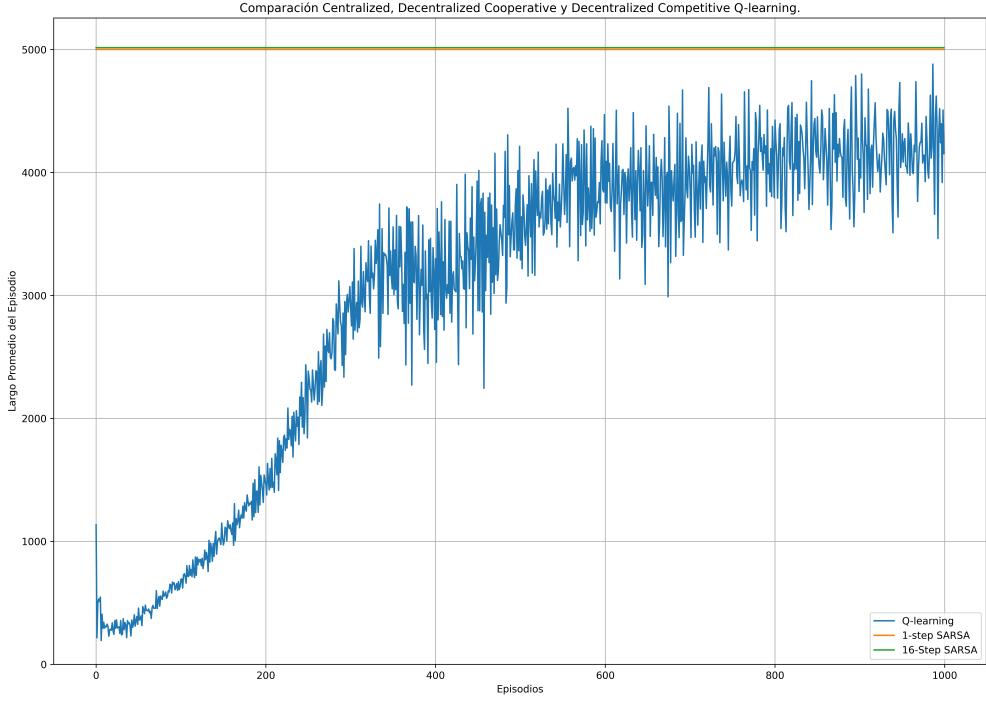


Figure 4: InvisibleDoorEnv SARSA, 16-step SARSA y Q-learning: Sin memoria

La Figura 5 contiene los resultados de SARSA, 16-step SARSA y Q-learning en el InvisibleDoorEnv, pero esta vez con 2-order memory. Con esta memoria, se pueden observar mejoras, por ejemplo, 16-step SARSA logra aprender en su totalidad el problema, y SARSA y Q-learning obtienen mejores resultados que antes, aunque de todas formas son resultados que no pueden ser considerados buenos, ya que empeoran a lo largo del tiempo.

Si la métrica más relevante es que el agente llegue al objetivo, entonces estos resultados demuestran que 2-order memory es suficiente por sí solo para resolver el problema, ya que todos los algoritmos logran finalizar los episodios en menos de el límite de steps. Sin embargo, SARSA y Q-learning empeoran a lo largo del tiempo, lo cual no es deseable, y 16-step SARSA tiene éxito porque su mecanismo de actualización compensa la memoria a corto plazo.

Esto es debido a las funciones de actualización de los valores Q que tienen estos algoritmos. Aunque ahora tengan una memoria de  $k=2$ , SARSA y Q-learning siguen actualizando sus valores a través de un solo paso hacia adelante, por lo que al alejarse dos estados del botón, ya olvidan que para haber abierto la puerta tuvieron que apretar el botón primero. A diferencia de lo anterior, 16-step SARSA, al actualizar sus valores, logra conectar el hecho de que apretó el botón con que, en menos de 16 pasos, logró pasar la puerta y llegar al objetivo.

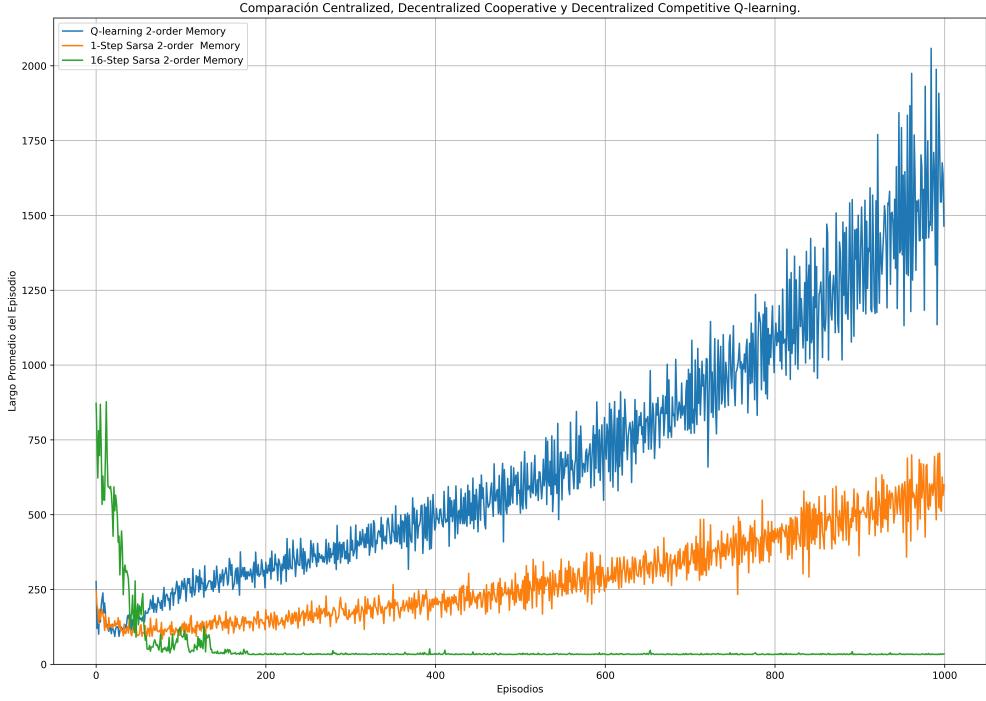


Figure 5: InvisibleDoorEnv SARSA, 16-step SARSA y Q-learning: 2-order memory

La Figura 6 contiene los resultados de SARSA, 16-step SARSA y Q-learning en el InvisibleDoorEnv, pero esta vez con 1-bit binary memory. En comparación con la Figura 4, sigue habiendo mejoras, sin embargo, claramente se obtienen resultados más inestables que en la Figura 5. En específico, Q-learning y SARSA muestran un rendimiento que empeora con el tiempo. Por otro lado, 16-step SARSA logra solucionar el problema, pero es mucho más inestable que antes, de hecho, en un momento olvida la política óptima.

Empíricamente, esta memoria es suficiente para resolver el problema, ya que ningún algoritmo llega al límite de steps. Sin embargo, los resultados muestran características indeseables, por ejemplo, SARSA y Q-learning empeoran a lo largo de los episodios, y 16-step SARSA, aunque a veces tiene éxito, es muy inestable.

La razón de este comportamiento es similar a la del experimento anterior. SARSA y Q-learning, con sus funciones de actualización de 1 paso, siguen "mirando" muy poco hacia adelante y no logran aprender la política de cómo usar el bit. 16-step SARSA logra aprender la política óptima, pero en general es más inestable y de hecho se le olvida en un momento. Esto ocurre debido a que su comportamiento epsilon-greedy podría actualizar el valor de un estado basándose en un resultado malo, lo que cambiaria para mal la política óptima aprendida, provocando que se le olvide la solución.

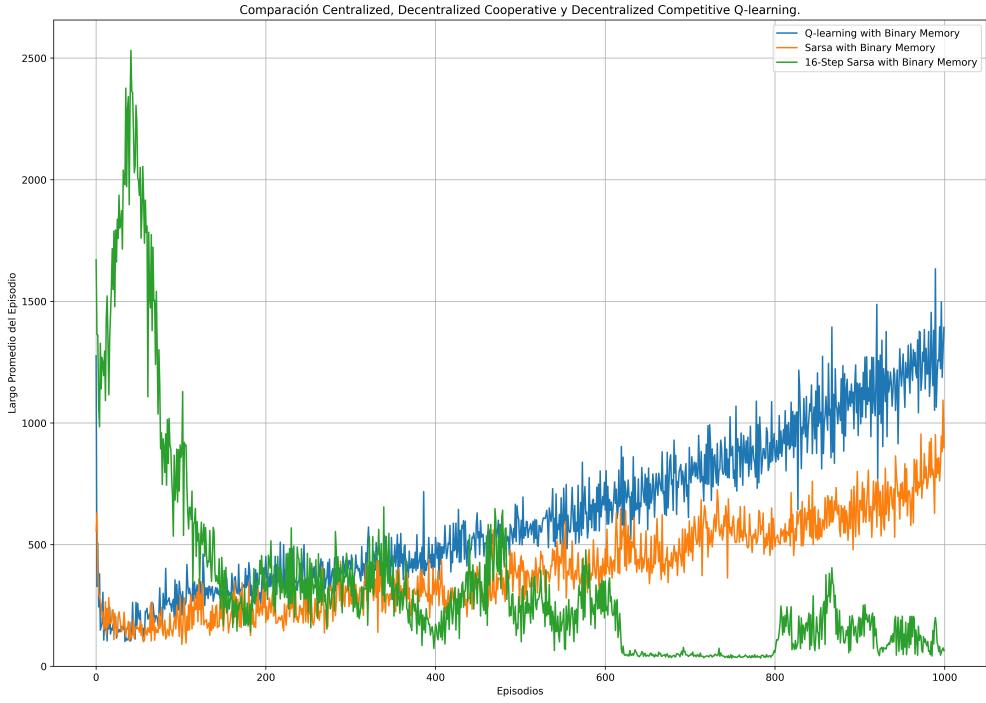


Figure 6: InvisibleDoorEnv SARSA, 16-step SARSA y Q-learning: 1-bit binary memory

La Figura 7 contiene los resultados de SARSA, 16-step SARSA y Q-learning en el InvisibleDoorEnv, pero esta vez con la variante de la k-order memory, donde el agente decide si quiere guardar alguna observación en particular. En comparación con todos los resultados anteriores, con este tipo de memoria obtenemos de manera definitiva los mejores resultados, ya que todos los algoritmos logran solucionar el problema y se eliminan las características indeseables que teníamos antes. Por ejemplo, ya no hay tanta inestabilidad, y SARSA y Q-learning no empeoran a lo largo del tiempo. También es notable que 16-step SARSA sigue siendo el algoritmo con el mejor rendimiento, especialmente en los primeros episodios, pero después de esto, todos logran aprender la política óptima.

Esta memoria es claramente suficiente para resolver el problema, principalmente debido a que el agente puede decidir sobre qué observación recuerda, a diferencia de los métodos anteriores donde la memoria se actualizaba de manera automática.

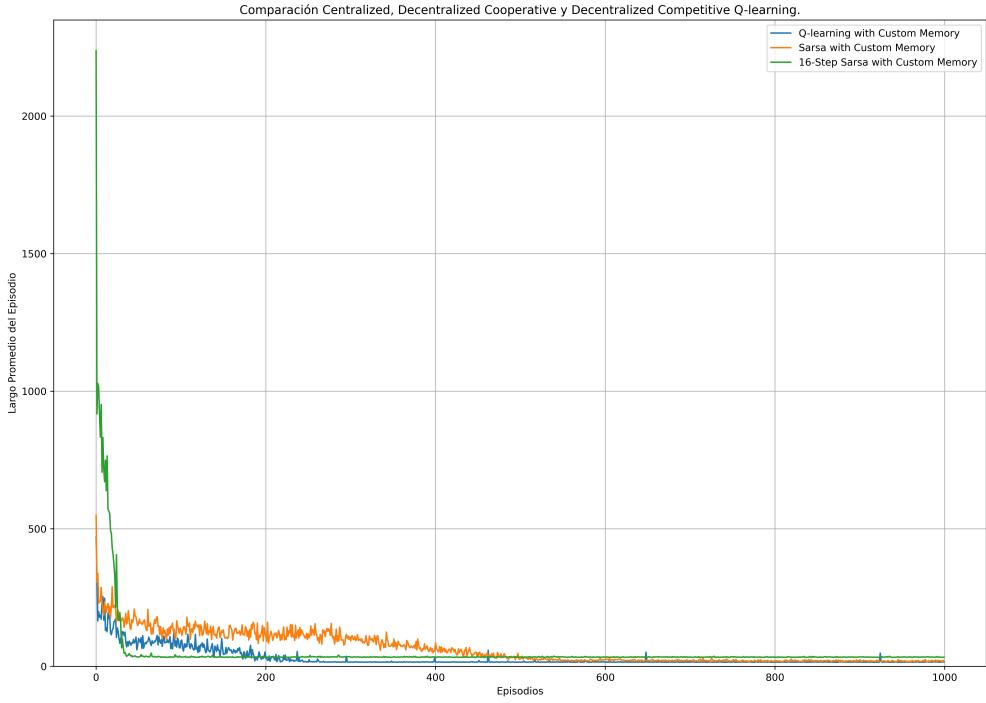


Figure 7: InvisibleDoorEnv SARSA, 16-step SARSA y Q-learning: k-order memory buffer variant

Para finalizar, considerando todos los resultados anteriores, el mejor tipo de memoria fue claramente la última. Empíricamente, esta memoria ofrece los mejores resultados, ya que todos los algoritmos logran aprender la política óptima y se eliminan los problemas indeseables de los experimentos previos. Desde un punto de vista más teórico, tiene mucho sentido que el agente pueda decidir qué observaciones memorizar para poder resolver mejor el problema.

En cuanto al mejor algoritmo, no diría que hay un claro ganador, ya que el rendimiento depende de la presencia de algún tipo de memoria, ósea que no es dependiente del algoritmo en sí (SARSA o Q-learning). Por ejemplo, cuando no hay memoria, Q-learning obtuvo mejores resultados que 1-step SARSA y 16-step SARSA. Sin embargo, en los experimentos con memoria, ambos SARSA demostraron ser mejores que Q-learning para obtener mejores resultados (Excepto en el último de manera leve).

En conclusión, para este tipo de problemas en donde no toda la información se encuentra disponible para el agente, la presencia de memoria es clave, por lo que, suponiendo que el agente si tiene algún tipo de mecanismo de memoria, entonces personalmente diría que SARSA, en específico SARSA con hartos steps (Ex. 16 steps) es el mejor algoritmo.