Universitat de les Illes Balears

Aprenentatge per reforç - SARSA

21722 - Intel·ligència Artificial

Khaoula Ikkene khaoula.ikkene1@estudiant.uib.cat
Adrían Ruiz Vidal adrian.ruiz2@estudiant.uib.cat

24 de novembre de 2024

Índex

1 Introducció

En aquesta pràctica se'ns presenta un exercici del tema d'aprenentatge per reforç. La nostra tasca consisteix a adaptar el codi proporcionat, que implementa un agent basat en l'algorisme Q-Learning, un mètode fora de política (off-policy), per convertir-lo en un agent que utilitzi l'algorisme SARSA, un enfocament de dins de política (on-policy).

2 Algorismes d'aprenentatge per reforç

Primer famre una petita introducció als algorismes d'aprenentatge. Aquest tipus d'algorismes són aquells que aprenen comportaments o polítiques òptimes mitjançant la interacció amb un entorn dinàmic.

Les característiques més rellevants dels algorismes d'aprenentatge per reforç inclouen la cerca per assaig i error i la recompensa retardada. L'agent intenta explotar les experiències prèvies, però al mateix temps busca explorar el desconegut per millorar les seleccions d'acció futures i obtenir la màxima recompensa.

La política és un concepte clau en el context dels algorismes d'aprenentatge per reforç. Defineix el comportament de l'agent en un moment determinat. Pot ser representada tant per una funció com per una taula de cerca. L'unic objectiu de l'agent és maximitzar la recompensa total, que ve definida pel model i les probabilitats de transició, que rep a llarg termini.

3 SARSA (State–Action–Reward–State–Action)

L'algorisme SARSA, al igual que el Q-learning, pertany a la categoria d'algorismes d'aprenentatge per reforç sense model, ja que no requereixen accés a la distribució de probabilitat de transició. Una de les característiques distintives de SARSA és que és un algorisme de dins de política (on-policy), la qual cosa implica que se centra a millorar la política de comportament actual utilitzada per l'agent per a prendre decisions. Això permet que SARSA aprengui el valor de la política realitzada per l'agent en cada moment. En canvi, els algorismes de fora de política (off-policy), com el Q-learning, permeten que l'agent aprengui a partir de les observacions d'una política òptima, fins i tot quan no la segueix directament.

Tant SARSA com Q-learning segueixen el patró d'iteració generalitzada de polítiques (Generalized Policy Iteration, GPI). Aquesta tècnica combina dos processos interactius i complementaris: l'avaluació de la política, que garanteix que la funció de valor sigui coherent amb la política actual, i la millora de la política, que ajusta aquesta política perquè sigui més cobdiciosa respecte a la funció de valor actual. Aquests processos s'alternen, i en el cas general, cadascun es completa abans de començar l'altre.

Aquest enfocament permet que ambdós algorismes es converteixin progressivament cap a una política òptima, tot i que difereixen en la manera com aprofiten i exploren el coneixement obtingut durant el procés d'aprenentatge.

4 SARSA VS Q-Learning

Donat que ambdós algorismes pertanyen a la mateixa categoria: algorismes d'aprenentatge per reforç, utilitzen el mateix enfocament: iteració generalitzada de polítiques, la seva estructura general és molt similar. No obstant això, les diferències es fan evidents en la manera

com cadascun d'ells actualitza els valors de les parelles (estat, acció), generant variacions lleugeres en aquest àmbit específic.

La següent figura il·lustra aquestes diferències, que explicarem en detall a continuació.

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figura 1: Diferències Q-Learning vs SARSA

Abans de capbussar-nos en l'explicació de l'estructura general del codi, aprofitarem per explicar alguns termes necessaris per a millor comprensió del codi.

- α: És el factor d'aprenentatge que determina fins a quin punt la informació nova reemplaça la informació antiga.
  Si $\alpha = 0$, l'agent no aprèn res, ja que no actualitza els valors de Q(s,a).
  Si $\alpha = 1$, l'agent confia únicament en la informació més recent, ignorant tota experiència anterior.
- γ: És el factor de descompte que control·la la importància de les recompenses futures respecte a les recompenses immediates.

  Si $\gamma = 0$, l'agent només considera les recompenses immediates.

  Si γ s'aproxima a 1, l'agent prioritza recompenses a llarg termini.
- ϵ : És el paràmetre d'exploració que control·la la proporció entre exploració (provar noves accions) i explotació (utilitzar allò que es coneix que és òptim).
- **Política basada en** $\epsilon - greedy$: És una tècnica alternativa a la selecció cobdiciosa. Amb probabilitat ϵ selecciona una acció aleatòria. Amb probabilitat $1 - \epsilon$ selecciona l'acció que maximitza Q(s,a). Al llarg dels episodis, ϵ sovint disminueix gradualment per a afavorir l'explotació cap al final de l'aprenentatge.

- **Convergencia** : Indica que els valors de Q(s,a) reflecteixen amb precisió les recompenses esperades sota la política π. Quan hi ha convergència, l'agent ha après a predir correctament el valor de les seves accions.

5 Adaptacions del Q-Learning

Basant-nos en la figura 1, explicarem les modificacions fetes que afecten principalment parts del mètode `train(self, discount, exploration_rate, learning_rate, episodes, stop_at_convergence=False)` de la classe `Agent`, i són els que comentarem a continuació.

Dins el bucle que itera per cada episodi, a part d'inicialitzar l'estat actual S, seleccionam una acció A per a l'estat S usant la política de ϵ - greedy:

```
if np.random.random() < exploration_rate:
    action = random.choice(self.environment.actions)
else:
    action = self.predict(state)
```

Aquest procés forma part de l'algorisme Q-Learning, on es trobava dins d'un bucle anidat que recorria cada pas de l'episodi. En SARSA, en canvi, es situa abans del bucle que itera per cada pas. El codi és el mateix, només s'ha reubicat segons el que especifica l'algorisme SARSA.

Dins el bucle que itera per a cada pas de l'episodi, aplicarem l'acció A a l'estat actual i obtindrem una recompensa R que s'afegirà al total de recompenses acumulades, així com l'estat següent al qual transicionarem, S'.

```
next_state, reward, status = self.environment._aplica(action)
cumulative_reward += reward
```

Aquest tros de codi no es modifica respecte a l'algorisme Q-Learning; simplement el canviem de posició, de manera que ara es converteix en la primera instrucció a executar dins el bucle anidat.

I ara triarem una acció A' de l'estat S' utilitzant la mateixa política ϵ-greedy:

```
if np.random.random() < exploration_rate:
   next_action = random.choice(self.environment.actions)
else:
   next_action = self.predict(next_state)
```

Aquest bloc de codi, com es pot veure, és similar al de triar una acció A de l'estat S e smentat anteriorment. L'únic canvi realitzat és que ara es representen l'acció següent (next action) i l'estat següent (next state), en lloc de l'acció i l'estat actuals.

Una vegada fet això, hem de comprovar que la parella (state, action) actual i següent, sigui una clau valida per poder usarla en la taula de Q:

```
if (
       state,
       action,
) not in self.Q.keys():  # ensure value exists for (state, action)
   # to avoid a KeyError
```

```
    self.Q[(state, action)] = 0.0


if (
        next_state,
        next_action,
) not in self.Q.keys():  # ensure value exists for (state, action)
    # to avoid a KeyError
    self.Q[(next_state, next_action)] = 0.0
```

Aquests blocs de codi s'han hagut d'afegir perque els valors no siguin nuls. El primer d'ells ja formava part de l'algorisme de Q-Learning mentre que el segon s'ha incorporat per assegurar que la parella de l'estat i l'acció següents tinguin valors no nuls.
Quan havíem modificat tot l'algorisme, aquesta era la part que mancava i per això no funcionava el programa correctament. Té sentit que, en calcular la següent acció a partir de l'estat amb SARSA, la parella no pugui tenir valors nuls.

I, igual que en el codi del Q-Learning, actualitzam el valor de la parella (state, action) d'acord amb la següent fórmula:

$$\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \left[R + \gamma Q(s',a') - Q(s,a)\right]} \tag{1}$$

D'aquesta sentència només hem de modificar la part de $\gamma Q(s', a')$. Ara, no necessitem calcular cap valor màxim, sinó que hem de prendre els valors corresponents al següent estat i la següent acció.

```
next_Q = self.Q[(next_state, next_action)]
self.Q[(state, action)] = self.Q[(state, action)] + learning_rate *
                          (reward + discount * next_Q - self.Q[(state, action)])
```

Una vegada pres aquest valor només hem d'incorporar-lo a la parella estat-acció actual. Finalment, s'actualitza l'estat actual i l'acció actual, en aquest cas del SARSA.

```
state = next_state
action = next_action
```

En el Q-Learning només s'actualitzava l'estat. Ara, com està definit a l'algorisme SARSA, també s'actualitza l'acció.

ajustos necessaris perqu'e el model de l'agent

Ara mostrarem els ajustos necessaris que hem fet per a assegurar que l'agent pugui adaptar-se a una nova posició inicial després de l'entrenament.

```
def reset(self):
        start_cell = (random.randint(0, 7), random.randint(0, 7))
        while not self.esValid(start_cell):
            start_cell = (random.randint(0, 7), random.randint(0, 7))
        self.__previous_cell = self.__current_cell = start_cell

        return start_cell
```

Les modificacions a la classe joc, dins el mètode reset(), permeten reiniciar l'agent a una posició aleatòria vàlida per evitar que sempre comenci des del mateix lloc. Aquest reset està

present en el mètode train(), de manera que a cada episodi l'agent calcula una nova posició inicial, ajudant-lo a aprendre a resoldre la cerca des de diferents punts d'inici.

```
def esValid(self, cell):
        new_cell = (cell[1],cell[0])
        return self.__maze[new_cell] == 0 and new_cell != self.__exit_cell and
        ↪  new_cell != self.__start_cell
```

Aquest mètode present dins reset() només comprova que la cel·la cercada no és ni una paret ni la posició inicial indicada ni la meta.

6 Medició de temps SARSA vs Q-Learning

Una vegada explicada les modificacions fetes en el codi, ens centrarem a explicar els diferents temps d'execució de cada algorisme per a un cert nombre d'episodis.

Per calcular el temps, només hem afegit un contador que s'inicialitza abans de la instrucció del mètode train i termina just després d'aquella instrucció. Els càlculs s'han fet per diversos nombres d'episodis: 100, 300, 500, 700, 900, 1000, 2000, 3000, 4000 i 5000. Hem dividit els valors en dues parts: la primera va de 100 a 900, variant cada 200 episodis; la segona va de 1000 a 5000, variant cada 1000 episodis.

Hem pres 5 mostres per cada nombre d'episodis i hem calculat la mitjana d'aquestes mostres. Per mostrar els resultats amb més detall, utilitzarem taules per representar els temps d'execució i gràfiques que reflectir la mitjana dels temps.

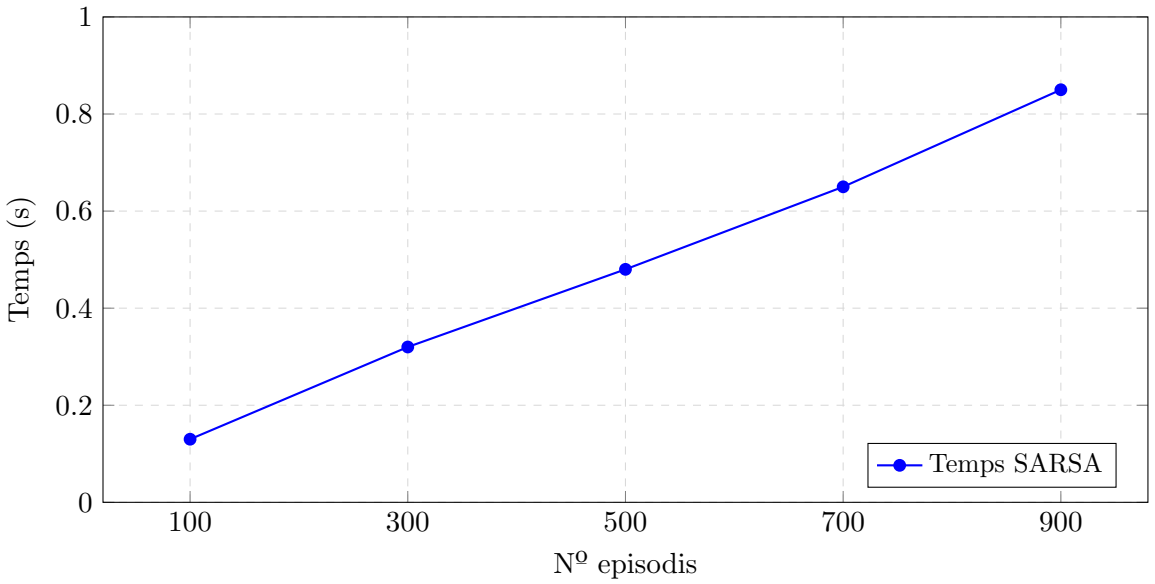
Cal comentar que totes les mostres mostrades han estat calculades a partir de l'estat inicial (0,0), assegurant així que totes les dades es generin sota les mateixes condicions i evitant que la posició d'inici afecti els resultats.

Primer començarem mostrant les dades i les gràfiques corresponents de l'algorisme SARSA:

Nº episodis	**Mostra 1**	**Mostra 2**	**Mostra 3**	**Mostra 4**	**Mostra 5**	**Mitjana**
100	0.12s	0.13s	0.13s	0.13s	0.13s	0.13s
300	0.31s	0.31s	0.31s	0.31s	0.32s	0.32s
500	0.47s	0.48s	0.48s	0.49s	0.48s	0.48s
700	0.67s	0.65s	0.64s	0.65s	0.66s	0.65s
900	0.88s	0.85s	0.82s	0.81s	0.90s	0.85s

Taula 1: Temps d'execució algorisme SARSA (Mides petites)

Els valors de la taula no varien molt amb les diferents mostres que feim. Els resultats per a cada nombre d'episodises es veuen molt regulars amb aquest algorisme.

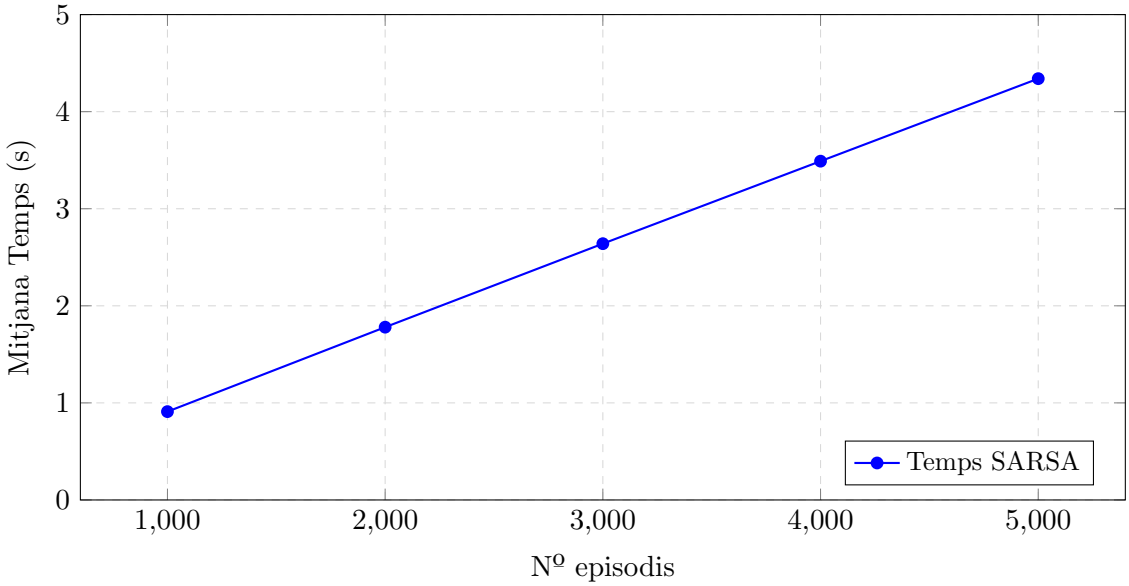


La gràfica mostra una evolució lineal progressiva a mesura que augmenta el nombre d'episodis, sense que es registrin grans variacions per a un nombre específic d'episodis.

Nº episodis	**Mostra 1**	**Mostra 2**	**Mostra 3**	**Mostra 4**	**Mostra 5**	**Mitjana**
1000	0.90s	0.91s	0.92s	0.93s	0.91s	0.91s
2000	1.77s	1.77s	1.77s	1.78s	1.80s	1.78s
3000	2.64s	2.64s	2.60s	2.64s	2.66s	2.64s
4000	3.53s	3.50s	3.48s	3.47s	3.48s	3.49s
5000	4.32s	4.32s	4.38s	4.35s	4.34s	4.34s

Taula 2: Temps d'execució algorisme SARSA

A aquesta taula veim resultats molt constants per a tenir un rang elevat d'episodis i podem veure com aquest algorisme és regular dins aquest àmbit.



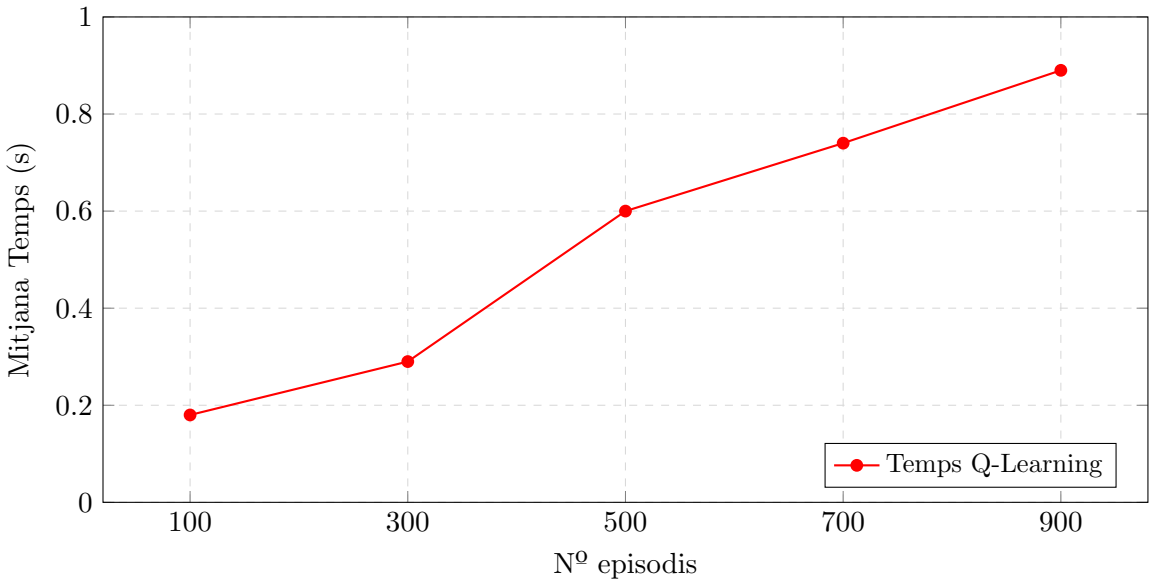
La gràfica presenta una línia gairebé recta, indicant la regularitat de l'algorisme, que es manté constant fins i tot amb un nombre elevat d'episodis.

A continuació, es presentaran les taules i les gràfiques corresponents per a l'algorisme Q-Learning.

Nº episodis	**Mostra 1**	**Mostra 2**	**Mostra 3**	**Mostra 4**	**Mostra 5**	**Mitjana**
100	0.11s	0.29s	0.12s	0.25s	0.11s	0.18s
300	0.29s	0.28s	0.30s	0.28s	0.28s	0.29s
500	0.49s	0.88s	0.48s	0.72s	0.45s	0.60s
700	0.73s	0.70s	0.94s	0.66s	0.66s	0.74s
900	0.96s	0.86s	0.85s	0.83s	0.95s	0.89s

Taula 3: Temps d'execució algorisme Q-Learning (Mides petites)

Amb la primera taula de l'algorisme Q-Learning, ja es poden observar diferències notables respecte a les taules anteriors. Les mostres presenten valors molt variables per al mateix nombre d'episodis, amb grans discrepàncies entre les diferents mostres.

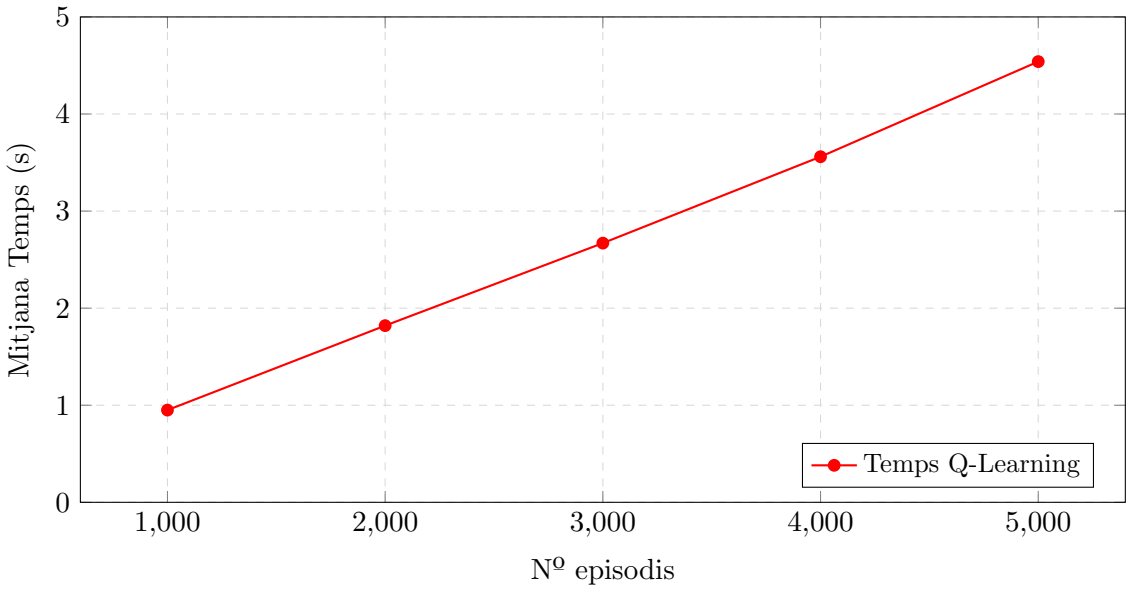


La gràfica reflecteix aquesta variabilitat, mostrant que l'algorisme Q-Learning és més inestable que SARSA. Per exemple, es pot observar que per a 300 episodis, la gràfica presenta una forma irregular.

Nº episodis	**Mostra 1**	**Mostra 2**	**Mostra 3**	**Mostra 4**	**Mostra 5**	**Mitjana**
1000	0.95s	0.94s	0.89s	1.03s	0.92s	0.95s
2000	1.95s	1.82s	1.75s	1.86s	1.73s	1.82s
3000	2.66s	2.76s	2.69s	2.59s	2.65s	2.67s
4000	3.61s	3.57s	3.48s	3.57s	3.61s	3.56s
5000	4.38s	4.86s	4.45s	4.40s	4.63s	4.54s

Taula 4: Temps d'execució algorisme Q-Learning (Mides grosses)

Aquesta taula revela que, tot i augmentar el nombre d'episodis, els resultats segueixen mostrant inconsistències, ja que per a diferents mostres hi ha valors dispars. Els temps d'execució respecte SARSA són una mica més lents però la diferència es molt petita.



La gràfica mostra que, amb un nombre més gran d'episodis, les dades es tornen més regulars a mesura que augmenta aquest nombre. Això implica que podríem predir el temps d'execució per a valors d'episodis com 6000 o 7000.

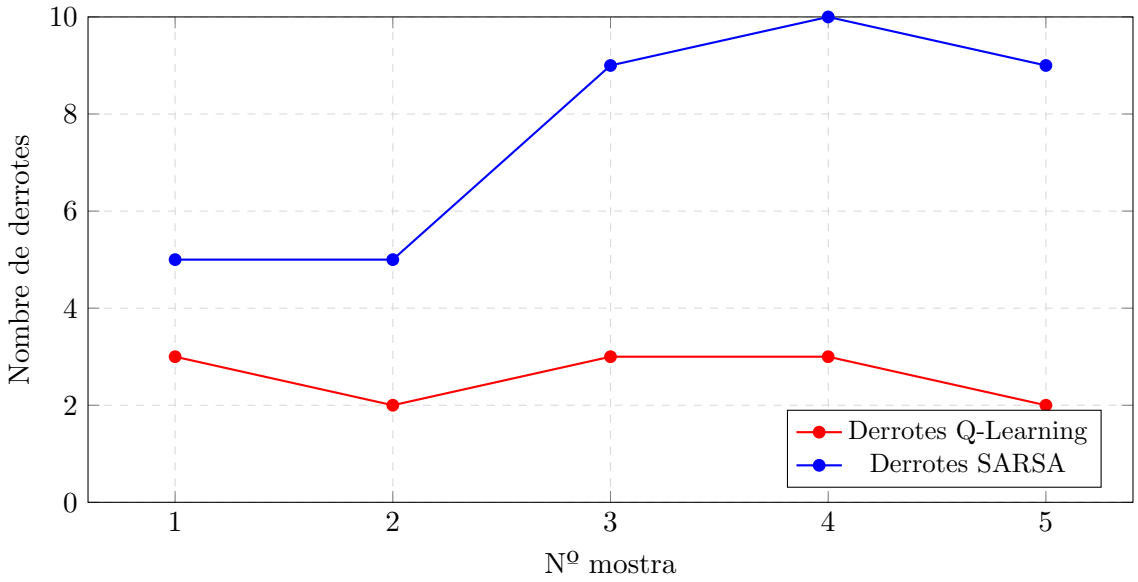
Ara presentarem un concepte important: la fiabilitat de cada algorisme. Aquest es mesura mitjançant el **nombre de derrotes** que experimenta cada algorisme, o dit d'una altra manera, el nombre d'episodis necessaris perquè l'algorisme s'estabilitzi, de manera que els resultats siguin únicament victòries, ja que l'agent ha après tot l'espai d'estats i sap quins moviments li permetran guanyar.

Això ho comprovarem amb 1000 episodis, com estava predeterminat. Tanmateix, hem realitzat proves amb diferents nombres d'episodis, i els resultats són gairebé idèntics, ja que amb qualsevol dels dos algorismes, dins un rang de 2 a 12 episodis, s'ha estabilitzat. Per tant, a continuació mostrarem la taula i la gràfica corresponent que compara el nombre de victòries de tots dos algorismes. Farem 5 mostres de cada algorisme per normalitzar els resultats.

Algorisme	**Mostra 1**	**Mostra 2**	**Mostra 3**	**Mostra 4**	**Mostra 5**
SARSA	5	5	9	10	9
Q-Learning	3	2	3	3	2

Taula 5: Nombre de derrotes SARSA vs Q-Learning

El nombre de derrotes de l'algorisme Q-Learning és bastant menor que SARSA, és a dir, necessita menys episodis per a estabilitzar-se i guanyar en els posteriors episodis.



7 Conclusió

Observant totes les taules i les gràfiques que hem vist, la diferència comuna que hi trobem és l'exploració de l'espai d'estats que realitza Q-Learning. Aquest algorisme explora més aquest espai en buscar el màxim, la qual cosa provoca la dispersió de les dades del temps. A més, en explorar més aquest espai d'estats per cada episodi, l'algorisme s'estabilitza més ràpidament pel que fa a la fiabilitat de trobar la solució òptima i, per tant, aconsegueix victòries en els primers episodis. En canvi, SARSA calcula l'actualització en funció de l'acció que segueix la política actual, la qual cosa redueix l'exploració dels estats i provoca variacions en els resultats en tots dos sentits.

Una diferència més irrellevant per la poca diferència que hi ha entre ambdos algoritmes és que SARSA és una mica més ràpid que el Q-Learning en aquests exemples, però no es gaira significativa.

En conclusió, els dos algorismes d'aprenentatge que hem vist, són algorismes rellevants dins aquest àmbit i que cadascú té les seves avantatges i els seus inconvenients. Q-Learning es preferible en situacions on l'exploració extensa és essencial per descobrir solucions òptimes, mentre que SARSA pot ser més adequat quan es prioritza un comportament més segur i consistent seguint la política actual.