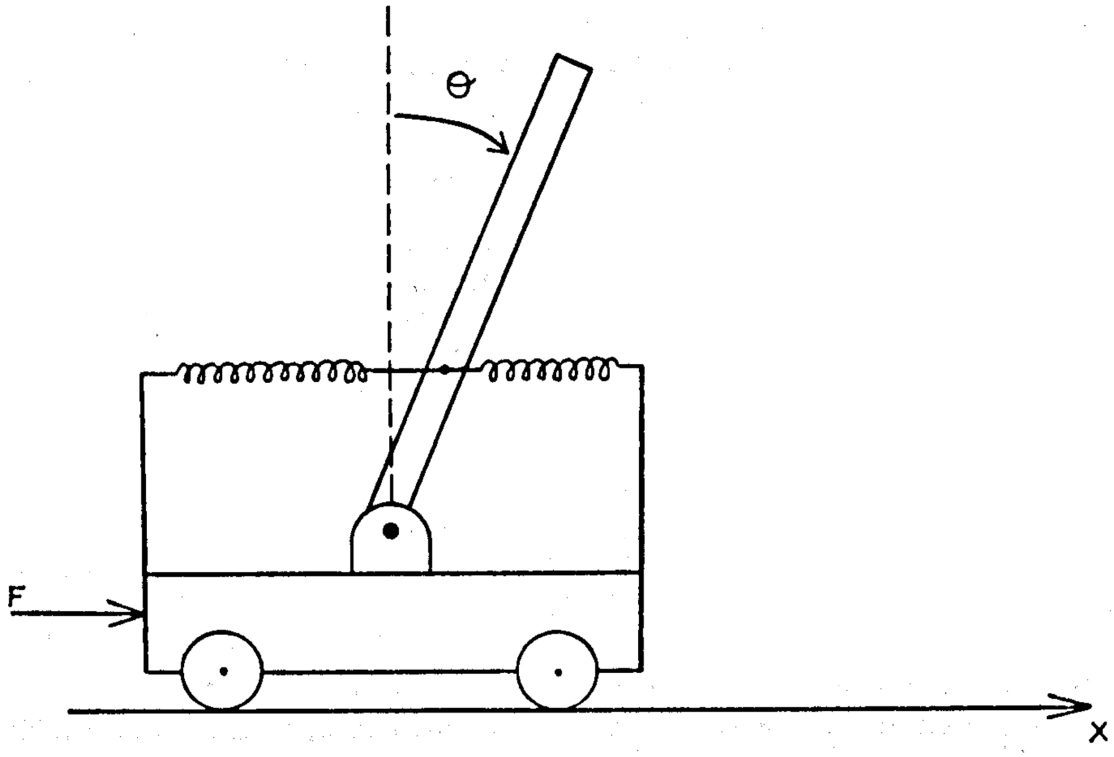
PRÁCTICA III

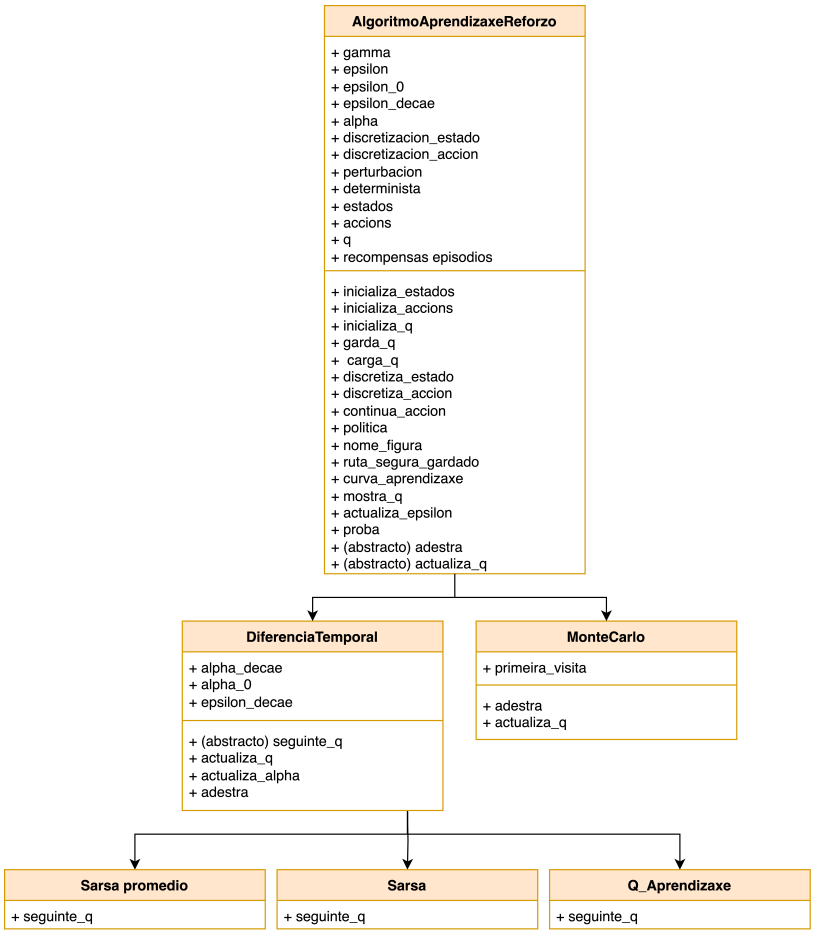
Aprendizaxe por reforzo

Pablo Chantada Saborido & José Romero Conde



1. Detalles de implementación

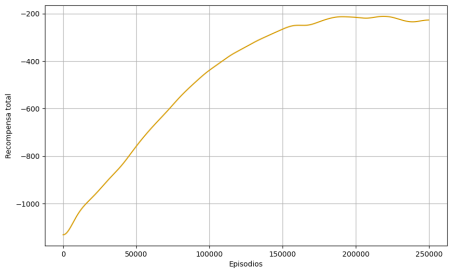
Motivados polo noso coñecemento teórico e a comprensión dos algoritmos non como secuencias de pasos que execútanse nun orden pero como principios teóricos que desembocan en implementacións; decidimos implementar a práctica dun xeito que reflictise as nosas creencias. Creiamos que, aínda que os algoritmos teñan propiedas moi distintas, son, polo menos dende algún punto de vista, moi similares tamén. Decidimos, entón, non implementar varias veces o mesmo código senón facelo o máis modular posibel. Foi tanta a sinxeleza que conseguimos, que aínda que o enunciado da práctica non pedise certas cousas (primeira visita ou non en MonteCarlo, inicialización informada, Sarsa Promedio ...), por completitude , foi moi doado conseguilo. Pódese ver abaixo un esquema do noso código. Donde unha clase abstracta `AlgoritmoAprendizaxeReforzo` define case todas as variables (arriba) e implementa case todos os métodos (abaixo) necesarios. Daquela `MonteCarlo` xa ten todo o que precisa para heredar e definir a súa clase. No caso dos algoritmos baseados en diferencias temporais, fixemos outra clase abstracta, que non implementando ningun algoritmo en concreto, só precisa tres liñas de Python por algoritmo particular que quera implementarse (e dicir: Sarsa, SarsaPromedio e Q-Aprendizaxe ocupan nove liñas en total). Isto foi posibel a partires da observación de que estes últimos só diferen no xeito de actualizar $Q(S, A)$.



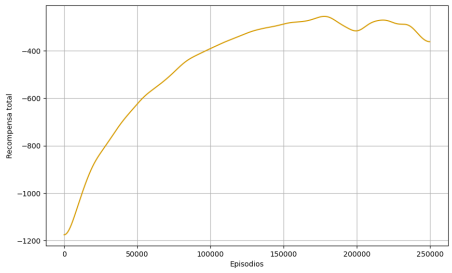
2. Hiperparámetros

Aínda que o problema a resolver é sixelo, a priori non sabemos (sobretodo sen experiencia) que hiperparámetros van resultar oportunos ou útiles, por tanto tentamos definir o maior número de hiperparámetros posibel; deste xeito, unha vez implementados só era cuestión de probar. Os hiperparámetros considerados e implementados ó largo do proceso foron:

- **Número de iteracións.** Especialmente importante a súa combinación ca finura ou *resolución* da discretización e cos decaementos doutros parámetros. Nunha das primeiras implementacións deixamos executando os tres algoritmos con 1M iteracións pero os resultados non pagan a pena (principalmente dado o alcance e obxectivo da práctica). Finalmente nos conformamos con 250K, que supón uns 6 minutos de adestramento, o cal é (pensamos) suficiente para obter resultados concluíntes.
- **Número de pasos máximo.** Dende un primeiro momento deixamolo ó máximo porque en calqueira caso non eran episodios largos.
- **Discretización do estado e acción.** A representación do estado non tivo en conta a natureza circular do problema. Deste xeito a nosa táboa Q é *dispersa* (é dicir, moitas entradas permanecerán baleiras e inutilizadas). Nas probas que usamos máis finura de muestreo do espacio $(\cos(\theta), \sin(\theta), v) \subset \mathbb{R}^3$ (as presentadas aquí) contan de 30 valores posibles para magnitude, e como o estado é unha terna, máis a acción que é un único valor fan que a nosa táboa Q sexa un *tensor* $(30 \times 30 \times 30 \times 30)$. Inicialmente probamos (i) menor granularidade (ii) diferente finura para acción e estado. Os resultados suxiren que a configuración actual é millor. De todos xeitos o sentido común dí que máis finura sempre vai ofrecer polo menos resultados iguais ou millores. A nosa elección, por tanto, ten que ver co número de episodios elixido.
- **Parámetro γ.** En todo momento fixamos $\gamma = 0.99$. Esta decisión respaldouse na unanimidade e consenso aparente cara este valor, despóis de buscar na Web. En calqueira caso con que sexa suficientemente alto é suficiente. Non ten, en cambio, porque ser igual a 1, de feito os episodios poderían ser infinitos.
- **Parámetro ε.** Cedo decatámosnos da importancia de este parámetro, motivo o cal decidimos implementar un decaemento (lineal) para que dentro dun aprendizaxe varie, acompañado de unha búsqueda *manual*. En calqueira caso, o importante e que nos primeiros episodios sexa o suficientemente explorativo (alto ε) e nos finais que non tambalee e opte sempre polas mellores accións (baixo ε), esto pódese apreciar claramente co caso do SarsaPromedio. Aquí ó ter implementado o decaemento, empezar con un ε alto significa ter a oportunidade de explorar máis opcións (potencialemte as óptimas). En ambas figuras, no final de todo $\varepsilon = 0.001$.



SarsaPromedio con $\varepsilon = 0.5$



SarsaPromedio con $\varepsilon = 0.2$

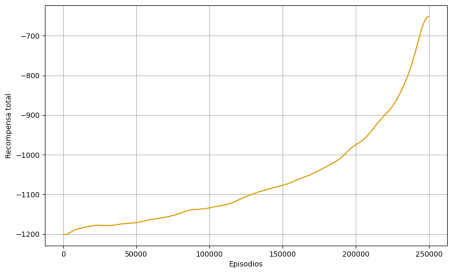
- **Parámetro α.** Aínda que no Sutton e Barto [1] propoñen unha versión do MonteCarlo que usa α dun xeito similar ós algoritmos de Diferencia Temporal:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

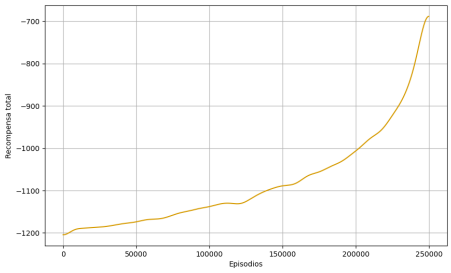
  nós restrinximos o noso uso deste parámetro ós segundos. O que podemos dicir é moi parecido ó discurso de ε, implementar un decaemento lineal foi clave para poder actualizar sen problema ó principio, pero asegurar a converxencia ó final. A nosa búsqueda non foi necesariamente exhaustiva porque cedo atopamos valores que proporcionaben bos resultados ($\alpha = 0.6$). Como particularidade, dicir que nun primeiro momento a implementación do decaemento era *de potencia*, pero como foi suxerido en clase, finalmente nos quedamos coa lineal.

- **Parámetro booleano, inicialización informada.** Como o problema era tan sinxelo pensabamos que inicializando dun xeito naïve a táboa Q, teríamos vantaxe. Na nosa implementación penalizamos o movemento (queremeos que esté parado arriba), e beneficiamos as posición altas e os movementos que fosen en contra da velocidade do péndulo. En xeral non observamos grandes diferencias, ou ben porque non foi afortunada a nosa implementación ou ben porque efectivamente o problema era tan sinxelo que apenas viuse afectado.

- **Parámetro booleano, primeira visita.** Malia que os cadernos vistos na asignatura só presentaron a versión primeira visita do método MonteCarlo, sabemos pola lectura [1] que existe a outra posibilidade e pareceunos suficientemente tentadora como para querer implementala. Pódese ver o (pouco) efecto nas figuras de abaixo (os resultados en proba para estos experimentos pódense na Táboa 1). Vese que con primeira visita, cando chega a 200K episodios xa ten a Recompensa total $G > 1000$ e non é certo no outro caso. Certamente a diferencia é minúscula (sabemos que asintóticamente non hai diferencia!) non obstante en adestramentos de moitos menos episodios sí pódese observarse algunha diferencia. En calqueira caso, ó poder dar a volta o pendulo, quizáis non sería esperable obter nunca moita diferencia, por esa propiedade cíclica.



MonteCarlo con primeira visita



MonteCarlo sen primeira visita

- **Parámetro booleano, decaementos.** Como adelantamos nas seccións de ε e de α, facer reducir os valores desos parametros co tempo ou non, sí entraña diferencia. Experimentalmente os resultados foron tan claros que xa cedo decidimos facer decaemento (en ambos parámetros).

Como definimos moitos hiperparámetros, naturalmente non podimos experimentar todas as combinacións, esto non foi un problema polos seguintes dous argumentos (i) o noso conocimiento teórico (aínda que por suposto, moi incompleto) peromitiunos dirixir a búsqueda e non probar cousas que ou ben non teñan sentido ou non iban mellorar os resultados (ii) o seren o problema tan sixelo, cedo acadamos con boas configuracións de hiperparámetros; poderiamos decir que, no espacio de hiperparámetros, para este problema, unha gran rexión conformaba eleccións aceptables de hiperparámetros.

3. Mellor política determinista

En canto a que algoritmo obtivo mellor política determinista, podemos dicir que Q Aprendizaxe, logo Sarsa, SarsaPromedio e por último MonteCarlo. Estos resultados obtivemolos do seguinte xeito: (i) entrenamos o necesario para xerar a Táboa 1 e de xeito automatico gardamos as respectivas táboas Q en arquivos `.npy` (ii) días despóis cargamos estas táboas e fixemos un script que probase 1000 veces cada algoritmo e fixese a media. Para comprobar a vericidade dos nosos resultados (e se querese replicar o experimento), engadimos a forma de execución que empregamos:

Reproducir a táboa de debaixo

```
$ python proba.py | grep media
```

Opcionalmente pódese cambiar as variables globales de devandito arquivo a (1, True) para ver cos ollos o comportamento do algoritmos.

Algoritmo	Recompensa en proba
Q Aprendizaxe	-149
Sarsa	-153
SarsaPromedio	-251
MonteCarlo	-597

Os resultados son bastante esperables. Como Q Aprendizaxe aproxima $q_\star$ (a táboa Q de *algunha* das políticas óptimas) en cada iteración en cal de q_{π_t} (a táboa Q da política no instante t), e o adestramento foi con episodios suficientes, e natural que sexa o que o faga mellor. Sarsa, en cambio también o fixo moi ben porque como comentamos antes asíntoticamente son iguais. Sabemos que non é sempre Q Aprendizaxe mellor a Sarsa pero para este problema (que tiña no caso no-perturbado ningunha componente estócastica) sí é o caso.

Parecería unha sorpresa que SarsaPromedio dese peores resultados que os outros dous. Non o é. O algoritmo bó é SarsaEsperado, que fai a media dos $Q(S', a)$ ponderandoos con $\pi_t(a|S')$, nos por simplificación o fixemos ponderandoos con $\frac{1}{|\mathcal{A}(S')|}$. É dicir, non implementamos correctamente a expectación. Non podemos dicir que a nosa implementación valga para dicir nada concluente do comportamento de SarsaEsperado, máis podemos dicir que nos valeu para observar que, en efecto os detalles importan. Tanto así que (como pódese ver na Táboa 1) baixo perturbacións ter en conta os $Q(S, A')$ para aquelas A' non elixidas, empeora drásticamente os resultados; máis que nun entorno estable.

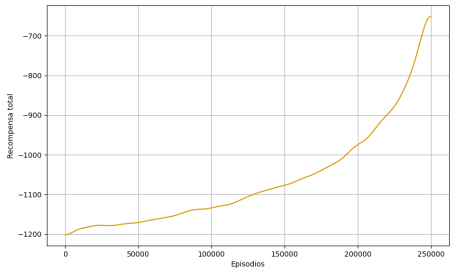
MonteCarlo, por outra banda queda nun segundo plano. Isto é porque MonteCarlo ten que esperar ó final de cada episodio para valorar e bondade das súas accións mentras que o resto de algoritmos o fan durante o mesmo episodio. Ademáis, MonteCarlo non sabe cales das accións que fixo nun mal episodio son malas nin cales das que fixo nun bo episodio son boas. É por isto que, para este problema, é esperable un peor rendimento por parte de MonteCarlo.

4. Mellor algoritmo de control

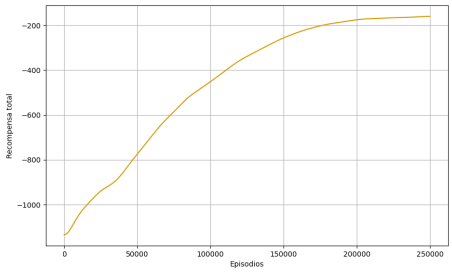
> For the control problem (finding an optimal policy, DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI). The differences in the methods are primarily differences in their approaches to the prediction problem.

Además da importancia que ten para un algoritmo comportarse ben unha vez está adestrado, é importante, sobretodo en Aprendizaxe por Reforzo, ofrecer bos comportamentos *mentras* aprende. Baixo os nosos experimentos, os mellores algoritmos para esta cuestión son os mesmos e no mesmo orden que para o exercicio anterior.

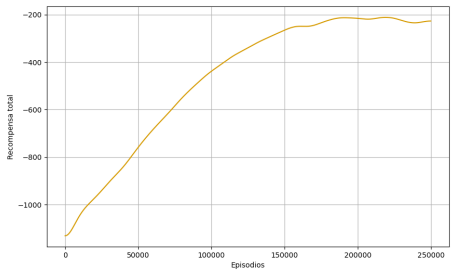
Se ben o problema de *mellor política determinista* redúcese a ver como o fan en proba (ou nos últimos episodios do adestramento se é que ε decae ate moi baixo), en cambio este problema de *mellor algoritmo de control* é relativo ó rendemento do algoritmo *ó largo dos episodios.* Abaixo pódese ver unha comparativa cos mellores resultados por algoritmo da táboa 1. Para mellor visulaización os resultados estan convolucionados cunha gausiana de $\sigma = \frac{\#episodios}{50}$. Independentemente do valor final obtido (o discutido no exercicio anterior) que verdaderamente é parecido o obtido nos tres métodos de Diferencia Temporal, pódese apreciar moi fácilmente as diferencias na velocidade de converxencia. Vemos que MonteCarlo necesita moitos episodios para comenzar a aprender significativamente e vemos que, dentro dos de Diferencia Temporal, Q Aprendizaxe e o que converxe máis rápido.



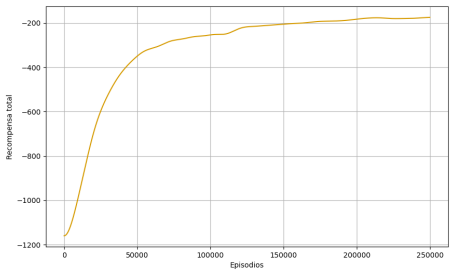
MonteCarlo



Sarsa



SarsaPromedio



Q Aprendizaxe

Malia estos resultados semellar satisfactorios, ó principio extrañounos que Q Aprendizaxe fora o mellor polo seguinte comentario de [1]:

> Although Q-learning actually learns the values of the optimal policy, its on-line performance is worse than that of Sarsa, which learns the roundabout policy. Of course, if ε were gradually reduced, then both methods would asymptotically converge to the optimal policy.

Entón, porque no noso experimento advertimos o contrario? Pensamos que débese a que o problema é tan sinxelo que ambos algoritmos só poden aprender o mesmo, é dicir, o óptimo. Entendemos que a circunstancia á que refirese o libro é dependente de entornos nos que existan solucións subóptimas ou triviais. Non é o caso co pendulo xa que non hai xeitos *mellores* de manter o péndulo arriba. Por tanto ambos queren aprender $\pi_\star$ (*algunha* política óptima), e Q Aprendizaxe ó facelo directamente é máis rápido.

5. Perturbacións

En canto ás perturbacións, planteamos os experimentos da seguinte forma: (i) adestrar o mesmo algoritmo con e sen perturbacións (como especificadas no enunciado da práctica) (ii) facer 5 episodios sen aprendizaxe, pero o que foi adestrado con perturbacións, tamén as terá en proba. Os resultados de este experimento, de novo, están na táboa 1. En canto as curvas de aprendizaxe, son consistentemente (nos catro algoritmos) moi parecidas a versión con e sen perturbacións. Non pensamos que as perturbacións sexan meritorias de máis disimilitude polos seguintes argumentos:

- Malia seren brutais no sentido de que a acción é extrema, ó ser con tan baixa probabilidad, non esperaríamos, de media, moita diferencia.
- En certa medida supón máis peso na parte de exploración e nese sentido as perturbacións son unha regularización.
- Aínda que as perturbacións están implementadas de xeito que enganan a percepción do axente (el pensa que fixo outra acción, e vai cambiar o $Q(S, a)$ de esa acción), só ocurre nun paso, no seguinte xa sabe onde está e (i) se aínda está a aprender, isto vale para facer máis exploración, (ii) se xa está aprendido, vai saber volver ó estado de inequilibro moi cedo. En calqueira caso, non hai moita disparidade.

6. Conclusións

Os experimentos demostran que Q Aprendizaxe supera outros algoritmos tanto en calidade de políticas deterministas como en velocidade de converxencia, o que atribuese á súa aproximación directa da función acción-valor óptima. Sarsa e SarsaPromedio séguenlle de preto, mentres que Monte Carlo vai por detrás debido á súa natureza de aprendizaxe episódica. As perturbacións tiveron un impacto mínimo, o que suxire robustez nos deseños dos algoritmos. A implementación modular demostrou ser eficaz, permitindo extensións sen fisuras.

Disfrutamos moito facendo a práctica porque o escenario tan académico non entorpeceu o noso acercamento a este novo paradigma de Aprendizaxe ate o momento descoñecido para nos. Esperamos, por outra banda, seguir aprendendo Aprendizaxe por Reforzo e mesmo usar as técnicas que xa coñecemos de Aprendizaxe si-no-semi-Supervisado. Para este problema seguramente fora máis axeitado unha representación non tabular do estado, porque baixo a nosa representación, todo o que aprenda dunha posición do pendulo dada con unha velocidade dada, non pode extrapolarlo a posicións e velocidades parecidas.

Anexo A: Experimentos

Malia ter feito experimentos preliminares, os resultados que narranse na práctica, as figuras obtidas, e mesmo os arquivos `.npy` foron todos obtidos nunha única execución.

Reproducir a táboa de debaixo

```
$ chmod 777 ./script
$ ./script
$ python parser.py resultados.txt
```

O *script* tarda entre tres e catro horas en executarse e xenera as figuras e os arquivos `.npy`, tamén xera un arquivo cos resultados chamado `resultados.txt`, se parsease como indicado, obterase a táboa en LaTeXdo mesmo xeito que o temos feito nos.

Algoritmo	d_e	d_a	Perturbación	γ	α	ϵ	Primeira Visita	Recompensa en proba
MonteCarlo	30	30	False	00.99	-	00.5	False	-504.88
MonteCarlo	30	30	False	00.99	-	00.5	True	-417.87
Sarsa	30	30	False	00.99	00.6	00.5	-	-98.71
Sarsa	30	30	False	00.99	00.2	00.2	-	-98.69
SarsaPromedio	30	30	False	00.99	00.6	00.5	-	-121.81
SarsaPromedio	30	30	False	00.99	00.2	00.2	-	-397.48
Q Aprendizaxe	30	30	False	00.99	00.6	00.5	-	-132.64
Q Aprendizaxe	30	30	False	00.99	00.2	00.2	-	-99.45
MonteCarlo	30	30	True	00.99	-	00.5	True	-585.19
Sarsa	30	30	True	00.99	00.6	00.5	-	-120.90
SarsaPromedio	30	30	True	00.99	00.6	00.5	-	-361.70
Q Aprendizaxe	30	30	True	00.99	00.6	00.5	-	-141.05

Táboa 1: Comparación de hiperparámetros e algoritmos.

A columna Recompensa en Proba é a media de 5 episodios sen aprendizaxe baixo as mesmas condicións de perturbación ou non que o adestramento.

Referencias

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.