Algorismes d'aprenentatge i optimització

Implementació i anàlisi
Novembre 2025

Autors:

Alejandro Martínez Hermosa
Martín Serra Rubio

Índex

1 Introducció

Per a la realització d'aquesta anàlisi, hem hagut d'implementar els algorismes d'aprenentatge per reforç Q-Learning, Sarsa, Programació Dinàmica i Montecarlo i un algorisme genètic en el joc de *Frozen Lake*, creat per OpenAI.

2 Explicació del joc

El joc és molt senzill. Donat un tauler de dimensions $n \times m$ s'ha d'aconseguir que l'agent trobi un camí des de la seva casella (marcada com una cadira) fins a la casella objectiu (marcada amb un regal). Si cau per un dels llacs congelats o supera/arriba a fer 100 passos en un mapa 4×4 o 200 passos en un mapa 8×8, l'agent perd.

Per poder arribar a l'objectiu, l'agent es pot moure en qualsevol de les 4 possibles direccions, representat directament amb un nombre enter per a cada direcció del 0 al 3, tot i que, pel fet que el paràmetre `is_slippery` està activat, l'agent tindrà certa probabilitat de realitzar l'acció en qualsevol de les altres dues direccions paral·leles. Aquestes dues accions són equiprobables.

Per saber en quina casella està, en cada moviment s'indica quina és la posició de l'agent amb un nombre enter representat per la següent fórmula:

Fórmula de posició

$$\text{posició\_actual} = \text{fila\_actual} \cdot \text{nombre\_columnes} + \text{columna\_actual}$$

Aquesta fórmula només representa que cada casella té un nombre enter de la forma que s'indica a la imatge a continuació:

Pel que fa a la recompensa obtinguda, l'agent obtindrà una recompensa de +1 en cas d'arribar a l'objectiu. En cas contrari, la recompensa obtinguda serà de 0.

3 Decisions de disseny

3.1 Algorismes d'aprenentatge per reforç

Primer hem generat la matriu de probabilitat d'estats inicialitzant totes les probabilitats a 0. A les files hem afegit els possibles estats, és a dir, totes les caselles del tauler, i a les columnes hem afegit els possibles moviments, és a dir, dreta, esquerra, amunt o avall.

La seva implementació ha estat bastant senzilla ja que, com he esmentat a l'apartat d'explicació del joc, els possibles estats estan numerats en ordre a partir del 0, igual que les accions, per la qual cosa ha estat molt senzill implementar això, ja que els índexs dels arrays que usem amb la llibreria `numpy` comencen per 0, és a dir, el primer element de l'array té l'índex 0.

3.1.1 Q-Learning

La modificació d'aquesta matriu s'ha fet mitjançant la següent fórmula, implementant així l'algorisme de Q-Learning:

Funció d'Actualització Q-Learning

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

Pel que fa a l'estructura del codi, seguim l'esquema clàssic de Q-Learning, mantenim una taula Q que s'actualitza després de cada interacció amb l'entorn. Implementem un bucle d'episodis on l'agent selecciona accions mitjançant una política ε-greedy i ajusta els valors Q usant l'equació d'actualització estàndard. A més, organitzem el programa en funcions separades per a la inicialització, l'actualització i la selecció d'accions, la qual cosa facilita la llegibilitat i la reutilització del codi.

3.1.2 Sarsa

La modificació d'aquesta matriu s'ha fet mitjançant la següent fórmula, implementant així l'algorisme de Sarsa:

Funció d'Actualització Sarsa

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

Pel que fa a l'estructura del codi, la implementació de SARSA segueix l'enfocament on-policy, actualitzant els valors Q a partir de les accions seleccionades per la pròpia política de l'agent. El bucle d'episodis integra l'elecció d'accions mitjançant una estratègia ε-greedy i l'actualització contínua de la taula Q, mantenint una organització modular per facilitar la claredat i l'extensibilitat de l'algorisme.

3.1.3 Monte Carlo

Pel que fa a l'estructura del codi, el mètode Montecarlo es basa en generar episodis complets i actualitzar els valors $Q(s, a)$ a partir del retorn mitjà de totes les visites. Implementem la política ε-greedy de manera incremental i emprem llistes de trajectòria per registrar transicions i calcular retorns, mantenint el codi organitzat en funcions independents.

Funció d'Actualització de Montecarlo

$$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$$

Es va implementar una estructura de dues fases: primer es genera i registra l'episodi complet en una llista temporal sota una política ϵ-greedy per garantir l'exploració, i després es fa una escombrada cap enrere per calcular el retorn acumulat (G). Seguint el pseudocodi, s'aplica la lògica First-Visit (verificant l'historial per ignorar estats repetits en el mateix episodi) i s'actualitza la matriu Q mitjançant una mitjana incremental (1/N) en lloc d'una taxa fixa, calculant així la mitjana exacta dels retorns obtinguts.

3.1.4 Programació dinàmica

Quan l'entorn té l'atribut `is_slippery=True`, les accions de l'agent es tornen imprevisibles: moure's en una direcció determinada no garanteix arribar a l'estat desitjat, sinó que l'agent té una probabilitat d'acabar en un estat diferent.

La programació dinàmica funciona assumint que el model de l'entorn és completament conegut i determinista, és a dir, que sabem amb seguretat quin estat s'aconseguirà després de cada acció. Quan les transicions no són deterministes, com passa amb `is_slippery=True`, això ja no funciona i fa que sigui molt difícil calcular la política òptima exacta, ja que no hi ha un camí garantit cap a la meta i la planificació basada en un model fix es complica molt.

3.2 Algorisme Genètic

El codi de l'algorisme genètic està organitzat en mòduls: inicialització de la població, avaluació, selecció, encreuament i mutació. Cada funció és independent, la qual cosa facilita ajustar paràmetres i reutilitzar el codi.

Cada individu és un array de longitud 16 que representa una política que seguirà l'agent, cada posició de l'array conté un nombre $i \in \{0, 1, 2, 3\}$ indicant el seu moviment segons l'estat.

Pel que fa a la funció fitness$(p : \text{política}, \text{episodis} : \text{int})$ l'únic que fa és que cada política "jugaün nombre *episodis* de vegades i s'avalua com de bona és dividint recompenses_totals/episodis.

Com a afegit extra, hem creat individus **elit**, aquests individus són els millors individus de la generació actual i els guardem per no perdre'ls. Amb aquest model individu/fitness, es repeteix el mateix procés *generacions* (paràmetre de la funció) vegades, cada generació selecciona els elits, avalua tots els individus, i crea una llista de fills seleccionant els pares segons la funció *fitness*, un cop amb els fills s'apliquen les mutacions a aquests i s'uneixen amb els elits obtenint la nova generació que serà sotmesa a aquest cicle una altra vegada.

4 Comparació de rendiments obtinguts

4.1 Q-Learning

Per analitzar el rendiment de Q-Learning, hem generat gràfics que mostren la probabilitat en tant per 1 d'èxit en cada episodi dels últims 100 episodis mesurats.

Com afecten els paràmetres a la gràfica?:

Paràmetres Base de l'Experiment

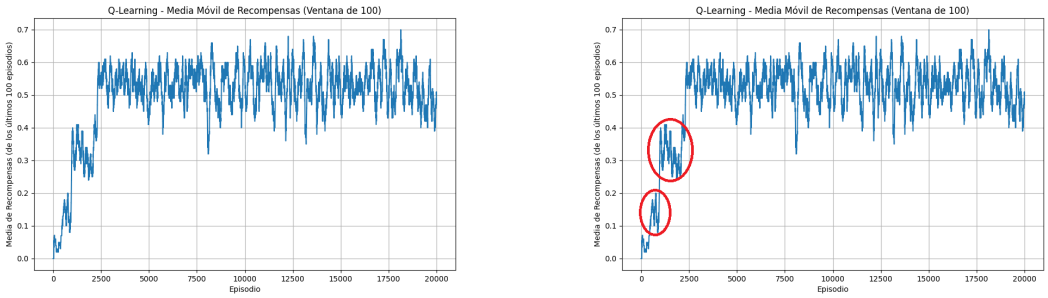
Com afecten els paràmetres?:

- α: és el valor que determina com de suau és la corba, aquest el que fa és donar més o menys importància a les dades noves que arriben a la matriu.
- γ: és el valor que determina com d'alta pot arribar a ser la corba, gràcies a aquest es tenen en compte les accions futures igual que les actuals.
- ϵ: és el valor que determina com de *greedy* és la política, un ϵ molt alt pot fer que no serveixi de res el que aprèn i triï sempre moviments aleatoris, mentre que un ϵ baix pot fer que trobi camins dolents com a solucions bones i s'estanqui, podria ser convenient tenir un ϵ_{decay} per poder anar modificant ϵ a mesura que l'agent aprèn, ja que a l'inici les dades no són gaire fiables i hauria de ser més *greedy*.
- *episodis*: és el paràmetre més senzill de veure, si hi ha pocs episodis la política no arribarà a aprendre suficient i es pot quedar en un màxim local o una zona dolenta, mentre que si hi ha massa execucions només allarguem innecessàriament ja que pot haver arribat ja al màxim.

Segons els diferents valors dels paràmetres α, γ, ϵ i *episodis* de la funció, s'obtenen gràfiques diferents:

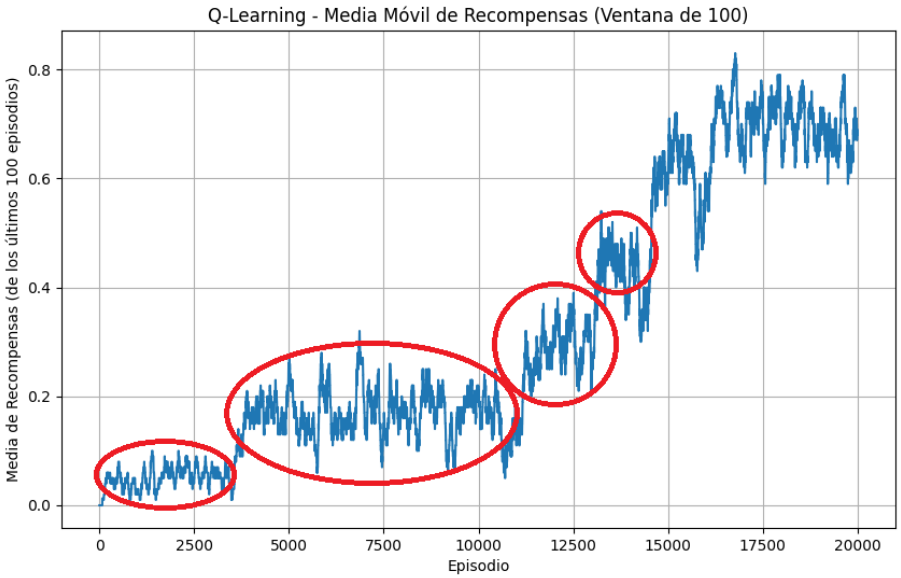
Aquest primer gràfic mostra una execució estàndard de 20.000 episodis.

Baseline Q-Learning ($\alpha = 0.1, \gamma = 0.99, \epsilon = 0.05$)



Anàlisi: Es veu un clar aprenentatge. Gràcies a $\epsilon = 0.05$ s'eviten estancaments en polítiques mediocres (cercles vermells), aquests estancaments podien ser màxims locals dels quals surt sent greedy.

Baseline Q-Learning ($\alpha = 0.05, \gamma = 0.99, \epsilon = 0.005$)

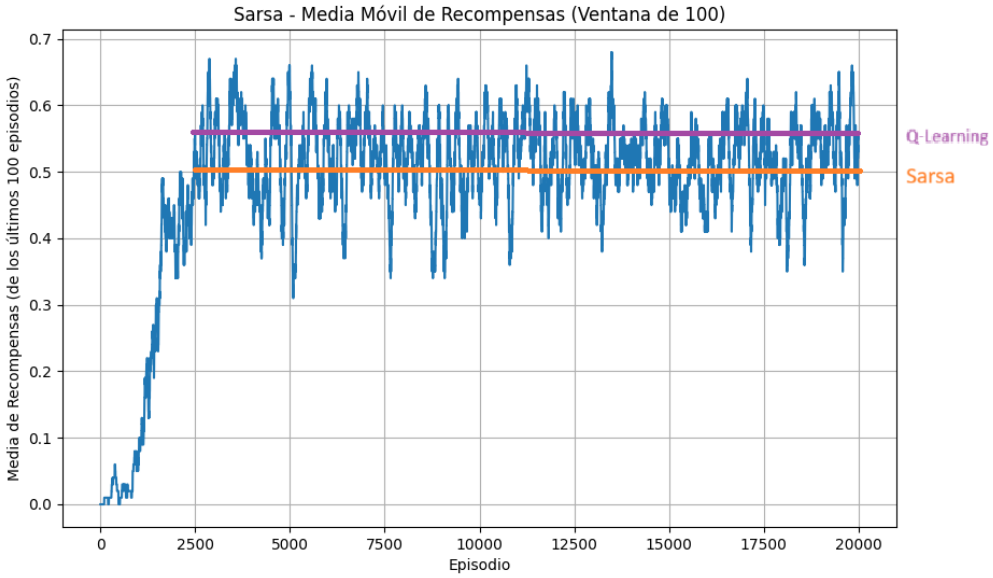


Anàlisi: I amb $\epsilon = 0.005$ li costa molt més sortir de màxims locals però quan perfecciona la política és millor.

4.2 Sarsa

Fent la mateixa execució que amb Q-Learning.

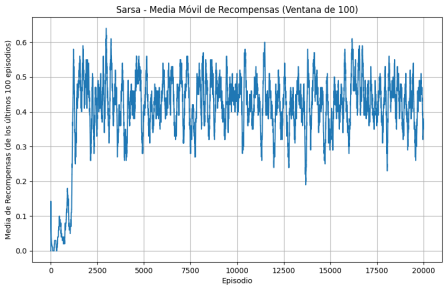
Comparativa: Sarsa vs Q-Learning ($\alpha = 0.1$, $\gamma = 0.99$ i $\epsilon = 0.05$)



Conclusió: L'única diferència tangible entre els algorismes és que ***Q-Learning* té una petita superioritat** quant a assegurar l'arribada a la victòria (té un comportament més *greedy* respecte als valors òptims), mentre que Sarsa es manté lleugerament per sota a causa de la seva naturalesa conservadora (*on-policy*).

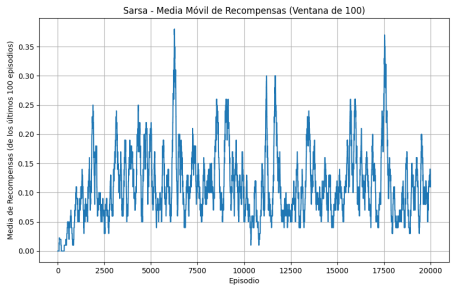
Jugant ara amb altres paràmetres, si canviem α passa el següent:

Efecte Alpha Alt ($\alpha = 0.3$)



Observació: S'aconsegueixen pics més dràstics i una gran inestabilitat.
Això passa perquè es dona massa valor a les dades noves, desestabilitzant el coneixement previ de l'agent.

Efecte Gamma Baix ($\gamma = 0.5$)

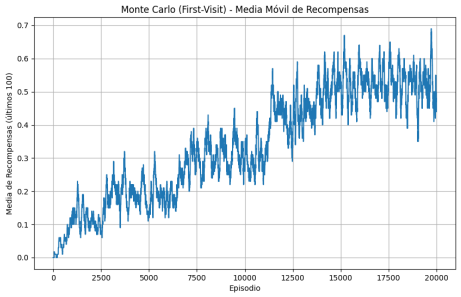


Observació: S'obté molt poca millora. Baixar la gamma fa que l'agent sigui "miop", ignorant les recompenses futures (el regal final) i centrant-se només en el pas immediat, que no dona punts.

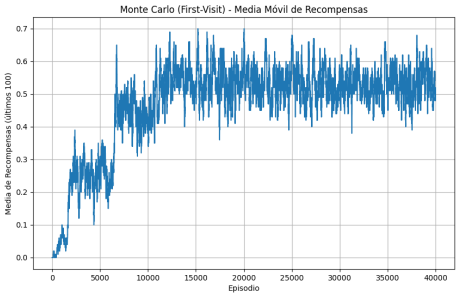
4.3 Monte Carlo

Fent ús de l'algorisme de Monte Carlo (First-Visit)

Evolució Monte Carlo: 20k vs 40k episodis ($\gamma = 0.99$ i $\epsilon = 0.05$)



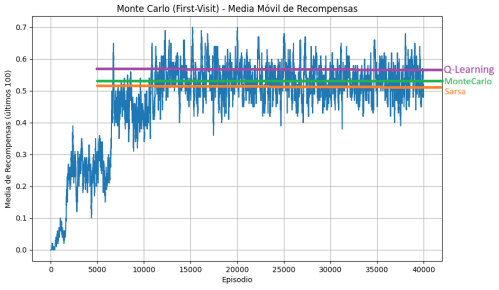
20.000 Episodis: Encara en fase d'aprenentatge.



40.000 Episodis: Estabilització aconseguida.

Monte Carlo és un algorisme que en general convergeix més lentament, això es deu al fet que per actualitzar ha d'esperar al final de cada episodi, al contrari que els altres dos algorismes que actualitzen en cada moviment fet. És per això que per obtenir un gràfic realment comparable amb els de *Q-Learning* i *Sarsa* hem de fer un entrenament d'episodis=40000.

Comparació al llarg del temps

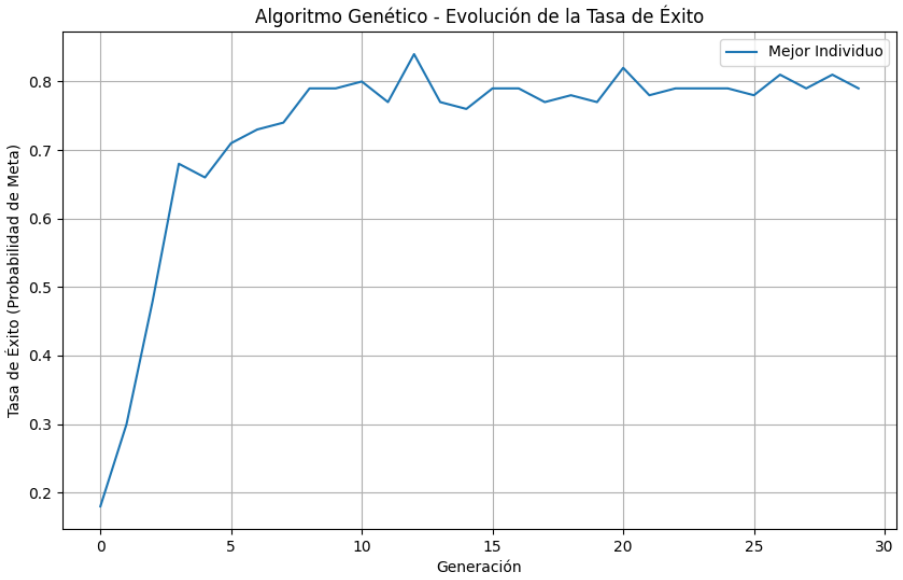


Observació: A llarg termini, Monte Carlo s'iguala a Sarsa i a Q-Learning.

4.4 Algorisme Genètic

En el cas de l'Algorisme Genètic, com que no té episodis sinó generacions, el gràfic és menys dens. Cada generació compta amb individus que s'avaluen i fusionen, si prenem la millor avaluació dels individus de cada generació obtenim el següent gràfic:

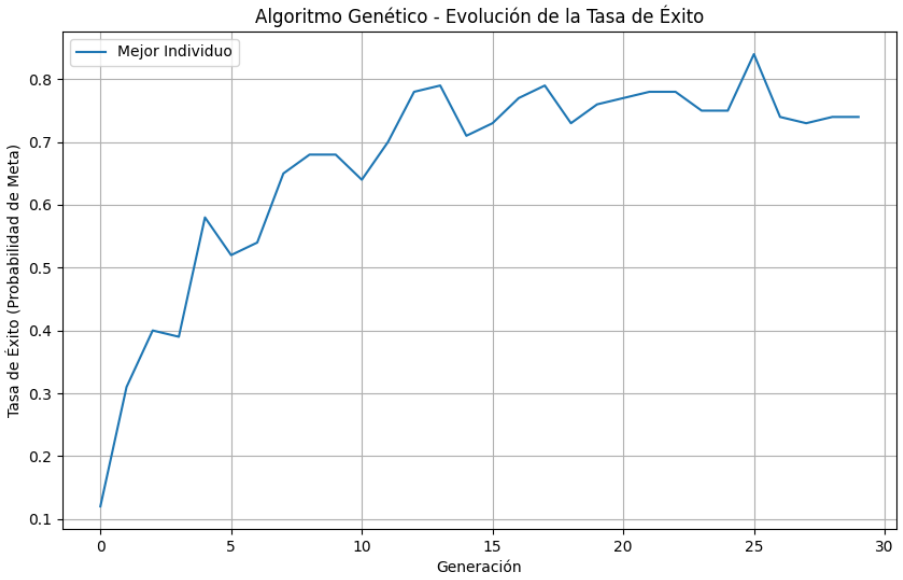
Algorisme Genètic (*generacions*=30,*tamano_poblacio*=100,mutacio=0.05,elits=5)



És realment tan bo com sembla?

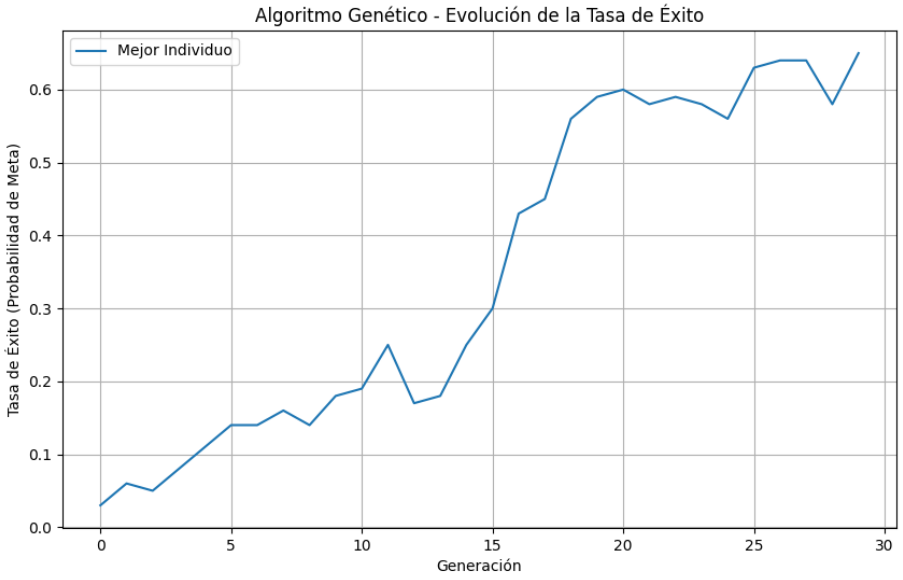
Aquest algorisme pot semblar molt prometedor, a les poques generacions ja puja a un valor molt alt igual o major que *Q-Learning*, el problema de l'algorisme genètic és la poca escalabilitat, el problema del *Frozen-Lake* amb un mapa 4x4 és molt petit.
Amb 30 generacions de 100 individus cadascuna ja triga una mitjana de 2 minuts d'entrenament
Per tant, en un mapa de 100x100 tindríem 10.000 estats, els nostres individus serien arrays de *length*=10.000 amb $policity[n] \in \{0,1,2,3\}$, la fusió entre polítiques probablement donaria polítiques dolentes i la funció $fitness$ seria inescalable, hauria d'executar cada individu moltes vegades.

Canvis en el paràmetre *mutació*



Anàlisi: Canviant el paràmetre *mutació* per un de més alt (=0.3) la gràfica canvia igual que si canviéssim el paràmetre ϵ en les altres, una pujada més inestable i un cim que varia més al llarg de les generacions.

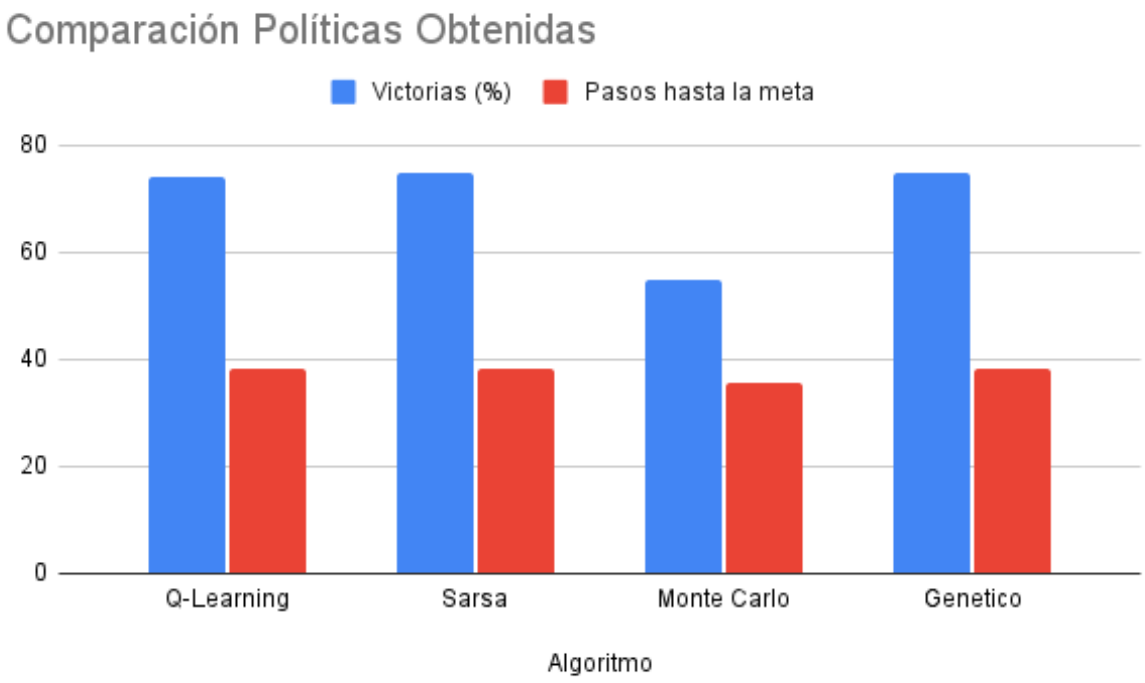
Reduir el paràmetre *tamano_poblacio*



Anàlisi: Seguint amb els paràmetres de l'inici, si ara canviem la quantitat de polítiques per generació (imposada a 100 de primeres per assegurar) a una quantitat més petita, els gràfics són inconsistents, depèn molt de la sort de les polítiques aleatòries inicials

4.5 Comparacions globals

Una vegada comparats els diferents algorismes durant l'entrenament, hem agafat de cadascun 15 polítiques entrenades i les hem avaluat, l'avaluació consisteix en executar-les 100 vegades amb un màxim de 100 *steps*, al final es fa la mitjana de vegades que arriba a la meta i els passos mitjans dels casos que han arribat a la meta:



Configuració Experimental

Aprenentatges per Reforç

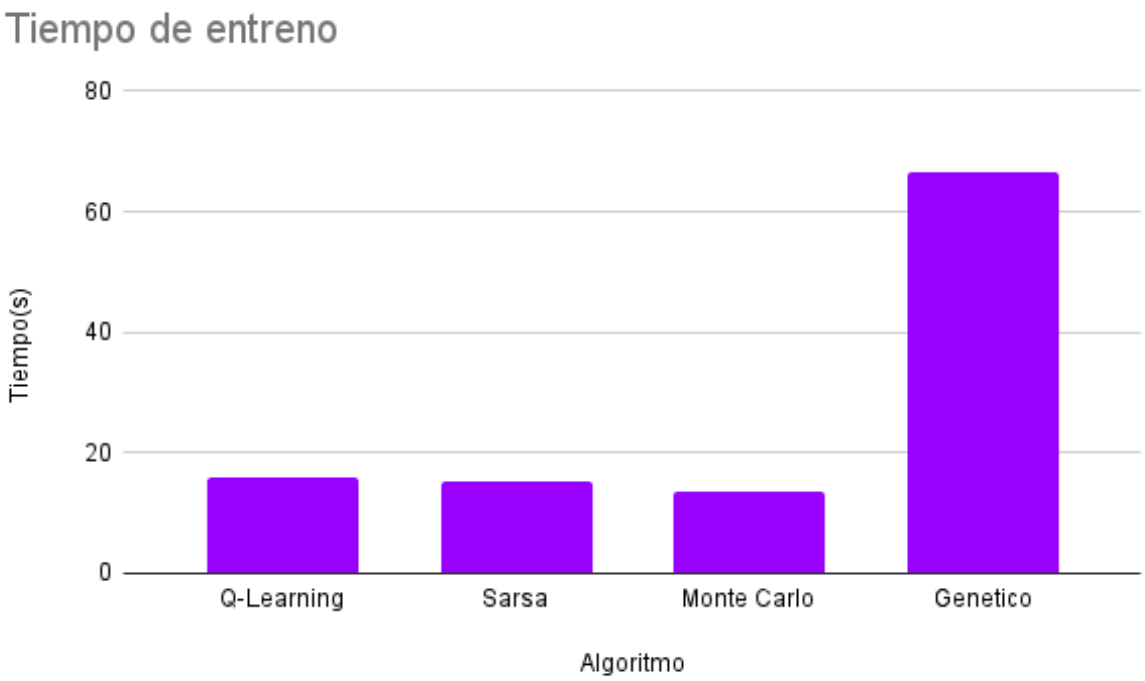
- **Episodis:** 20.000
- **Alpha (α):** 0.1
- **Gamma (γ):** 0.99
- **Epsilon (ϵ):** 0.05

Algorisme Genètic

- `tamano_poblacio`: 100
- `generacions`: 30
- `prob_mutacio`: 0.05
- `elits`: 5

Es veu una clara minoria de victòries usant Monte Carlo, això es deu al fet que creix molt més lentament que els altres algorismes, amb el doble d'episodis tindria el mateix % que els altres.

Una altra comparació que hem fet és el temps mitjà d'entrenament de cada algorisme amb els mateixos paràmetres que abans, sortint:



Totes tenen un temps semblant menys l'algorisme genètic, que es dispara a causa de la gran quantitat d'avaluacions que fa la funció *fitness* i totes les generacions creades per perfeccionar la política.

5 Anàlisi de les polítiques resultants

5.1 Sarsa

Sarsa és un algorisme *on-policy*, és a dir, aprèn utilitzant la mateixa política que s'emplea per actuar. Els indicadors principals són:

- **Recompensa acumulada:** mostra si l'agent aprèn a arribar a la meta. S'acostuma a utilitzar la mitjana de diversos episodis per suavitzar la variabilitat.
- **Durada dels episodis:** mesures de quants passos triga l'agent a completar un episodi; una durada menor amb recompensa alta indica una política més eficient.
- **Error de TD:** indica com de bé s'està ajustant la funció de valor; quan disminueix cap a zero, la política s'estabilitza.
- **Exploració i explotació:** el percentatge d'accions aleatòries disminueix a mesura que l'agent aprèn, mostrant estabilitat de la política.

5.2 Q-Learning

Q-Learning és *off-policy*, actualitza els valors assumint que l'agent sempre triarà la millor acció, independentment de la política d'exploració. Indicadors clau:

- **Recompensa acumulada i durada:** tendeix a créixer més ràpidament que Sarsa, encara que pot presentar oscil·lacions degudes a l'actualització agressiva.
- **Convergència de la taula Q:** mesura l'estabilitat dels valors de les accions; quan es manté estable, la política ha convergit.
- **Política final:** s'analitza la coherència de les accions òptimes i la taxa d'èxit en l'entorn.

5.3 Montecarlo

Els mètodes de *Montecarlo* calculen els valors d'acció a partir del retorn complet dels episodis. Indicadors rellevants:

- **Recompensa acumulada i durada:** similar als altres mètodes, encara que la variabilitat és més alta en episodis inicials.
- **Valor mitjà per estat-acció:** l'estimació es basa en la mitjana dels retorns observats; la política s'ajusta a mesura que s'acumulen més episodis.
- **Estabilitat de la política:** es mesura per la consistència dels valors i de les accions escollides, així com per la reducció de canvis en la política al llarg del temps.

5.4 Algorisme Genètic

- **Polítiques conservadores:** les polítiques generades tendeixen a ser més segures i menys arriscades, prioritzant arribar a la meta per sobre de l'eficiència.
- **Funció *fitness*:** avalua cada individu executant la política només 100 vegades i comptant quantes arriben a la meta, sense tenir en compte la longitud del camí.
- **Ajust de la política:** com a resultat, les accions seleccionades busquen maximitzar l'èxit de completar l'episodi, més que reduir el nombre de passos.

6 Bibliografia

- Documentació Gymnasium Frozen_Lake
- Documentació Gymnasium Env
- Documentació Gymnasium Discrete