Escape del Laberinto Mutante

501351-1 INTELIGENCIA ARTIFICIAL (S2-2025)

Javier Alejandro Campos Contreras
Antonio Jesus Benavides Puentes

Septiembre 2025

1. Explicaciones de los algoritmos

Todos los algoritmos descritos a continuación reciben como entrada el conjunto de movimientos válidos que pueden ejecutarse:

- Arriba.
- Abajo.
- Derecha.
- Izquierda.
- No Moverse.

El conjunto de movimientos válidos depende de la vecindad inmediata del jugador en ese momento; es decir, si alguna dirección está bloqueada por una muralla, dicha dirección no aparecerá en la lista resultante.

1.1. Random

Este algoritmo selecciona de manera aleatoria uno de los movimientos disponibles en cada turno.

1.2. Greedy

Este algoritmo identifica la meta más cercana y se dirige hacia ella siguiendo la trayectoria más corta, sin considerar la presencia de murallas en el camino.

Como consecuencia, es común que quede atrapado frente a obstáculos, esperando a que estos se muevan.

1.3. Learning Real-Time A* (LRTA*)

Este algoritmo surge como una variante del clásico A*, diseñada para entornos dinámicos. Se basa en la función tradicional $F = G + H$, donde G corresponde al costo acumulado para llegar a cada posición y H es la heurística estimada.

A diferencia de A*, que busca calcular rutas completas, LRTA* se centra en evaluar únicamente los movimientos más prometedores entre sus vecinos inmediatos. Gracias a este enfoque local, resulta especialmente adecuado para esquivar obstáculos y adaptarse a cambios en el entorno en tiempo real.

1.4. Q-Learning

Este algoritmo requiere un preentrenamiento para construir una Q-table, en la cual almacena el valor esperado de cada acción en cada casilla del laberinto.

Dado que el entorno es dinámico, la tabla no refleja soluciones exactas, sino patrones recurrentes que resultan útiles para la búsqueda. El agente alterna entre dos estrategias:

- **Explorar:** probar acciones nuevas para descubrir información adicional.
- **Explotar:** elegir las acciones que la Q-table ha identificado como más beneficiosas.

En consecuencia, Q-Learning es un enfoque de aprendizaje por refuerzo basado en la experiencia acumulada.

1.5. Q-Learning + LRTA*

Este algoritmo corresponde a una propuesta original, aunque probablemente ya exista en la literatura bajo otra denominación.

La motivación surge de una limitación observada en Q-Learning: aunque el agente aprende patrones efectivos a partir de la experiencia, no siempre considera de forma explícita la ubicación de la meta más cercana. Como resultado, incluso estando junto a una meta no visitada, puede ignorarla.

Para solucionar este problema, se propone una combinación de Q-Learning (basado en la experiencia pasada) con LRTA* (enfocado en el estado presente y la meta más cercana). La decisión final se obtiene mediante una suma ponderada de las estimaciones de ambos algoritmos. Por defecto, los pesos asignados a cada uno son de 0.5, equilibrando así la influencia de la experiencia y de la situación actual.

1.6. Algoritmo Genético

1.6.1. Cromosomas y Genes

Para este algoritmo se tomó como base el método *Q-Learning + LRTA**. El algoritmo genético implementado funciona de la siguiente manera: se genera una población de cromosomas, cada uno compuesto por los siguientes genes:

- **Alpha** ($\alpha \in [0,1]$): representa la tasa de aprendizaje de la Q-Table, es decir, qué tan rápido se actualiza el conocimiento previo con la nueva información. Valores cercanos a 1 implican un aprendizaje más rápido.
- **Gamma** ($\gamma \in [0,1]$): factor de descuento de recompensas futuras. Indica cuánto valoramos las recompensas futuras frente a las inmediatas. Valores cercanos a 0 priorizan las recompensas inmediatas.
- **Beta** ($\beta \in [0,1]$): pondera la importancia de la Q-Table sobre la heurística.
- **Omega** ($\omega \in [0,1]$): pondera la importancia de la heurística sobre la Q-Table.

Notas:

1. Los parámetros β y ω son complementarios, es decir, $\beta = 1 - \omega$ y $\omega = 1 - \beta$.
2. De manera similar, α y γ son complementarios, es decir, $\alpha = 1 - \gamma$ y $\gamma = 1 - \alpha$.

1.6.2. Fitness

En nuestro caso, la población inicial consta de 100 cromosomas. Cada cromosoma se evalúa en un laberinto y se calcula su *fitness* durante un número determinado de ticks, repitiendo el proceso a lo largo de 100 generaciones. La función de fitness está definida como:

$$\text{Fitness}(j) = \begin{cases} \text{bonus} + \dfrac{1}{1+t}, & \text{si } d = 0 \quad \text{(jugador alcanzó la meta)} \\ w_c \cdot \dfrac{1}{1+d} + w_e \cdot \dfrac{1}{1+t}, & \text{si } d > 0 \quad \text{(jugador no alcanzó la meta)} \end{cases} \tag{1}$$

donde:

$$\begin{aligned} d &= \text{distancia del jugador a la meta}, \\ t &= \text{cantidad de ticks consumidos}, \\ w_c &= \text{peso de cercanía}, \\ w_e &= \text{peso de eficiencia}, \\ \text{bonus} &= 2{,}0 \quad \text{(premio por alcanzar la meta)}. \end{aligned}$$

1.6.3. Crossover and Mutation

Una vez evaluados los cromosomas, se seleccionan los dos mejores y se realiza el *crossover* mediante la siguiente operación:

$$\text{media} = \frac{v_1 + v_2}{2}, \quad \delta = \frac{|v_1 - v_2|}{2} \tag{2}$$

$$v_{\text{hijo}} \sim [\text{media} - \delta, \text{media} + \delta], \quad v_{\text{hijo}} = \text{máx}(\text{min\_val}, \text{mín}(\text{max\_val}, v_{\text{hijo}})) \tag{3}$$

donde v_1, v_2 son los valores del gen de los padres, v_{hijo} es el valor del gen del hijo y *min_val*, *max_val* son los límites permitidos para el gen.

Para mantener diversidad en la población, se aplica mutación controlada en un 8 % de los cromosomas de cada generación:

$$\Delta v \sim [-v_{\text{original}}, 1 - v_{\text{original}}], \quad v_{\text{mutado}} = \begin{cases} v_{\text{original}} + \Delta v, & \text{con probabilidad } p_m \\ v_{\text{original}}, & \text{con probabilidad } 1 - p_m \end{cases} \tag{4}$$

donde $p_m = 0{,}08$ es la probabilidad de mutación.

Al finalizar todas las generaciones, se selecciona el cromosoma con mayor *fitness* como el parámetro óptimo para el algoritmo.

Al igual que *Q-Learning + LRTA**, la idea inicial surgió de nuestra propia propuesta. Sin embargo, al desconocer los valores ideales de los parámetros (α, γ, β, ω), decidimos implementar esta versión genética, permitiendo que el algoritmo encuentre automáticamente los valores que maximizan el rendimiento.

2. Información de la máquina y software en la que se corrieron los experimentos

2.1. Hardware y sistema operativo

Sistema Operativo: Ubuntu 24.04.3 LTS x86_64

Kernel: 6.14.0-32-generic

Shell: bash 5.2.21

Terminal: /dev/pts/0

CPU: Intel i5-8400 (6 núcleos) @ 4.0 GHz

GPU: NVIDIA GeForce RTX 2080

Memoria RAM: 16 GB

Virtualización: VT-x

Cachés:

Caché	Tamaño	Instancias
L1d	192 KiB	6
L1i	192 KiB	6
L2	1.5 MiB	6
L3	9 MiB	1

Cuadro 1: Resumen de cachés de la CPU

2.2. Software y proyecto Python

Python requerido: 3.12

Dependencias principales:

Paquete	Versión mínima
matplotlib	$\geq$ 3.10.6
questionary	$\geq$ 2.1.1
black	$\geq$ 25.1.0
isort	$\geq$ 6.0.1
pydocstyle	$\geq$ 6.3.0

Cuadro 2: Dependencias principales del proyecto

3. Resultados

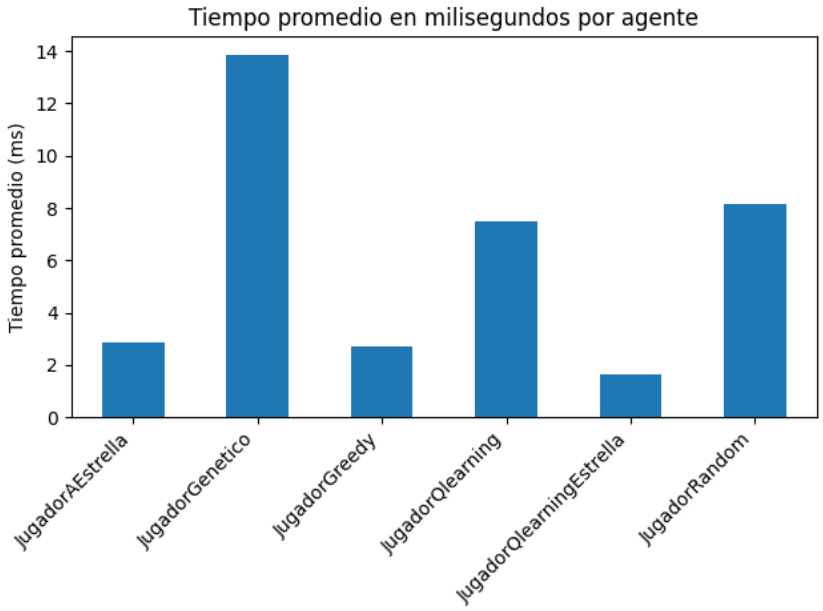

Figura 1: Tiempo promedio de ejecución por agente, medido en milisegundos.

Nota: En el caso del agente genético, el tiempo mostrado incluye tanto la fase de entrenamiento como la de resolución del problema.

Como se observa en la gráfica 1, los algoritmos basados en heurísticas presentaron tiempos considerablemente menores en comparación con los algoritmos de aprendizaje por refuerzo (Q-learning), el híbrido (Q-learning + LRTA*), los genéticos (variante genética de Q-learning + LRTA*) y el algoritmo aleatorio, lo cual se entiende dado que este último no sigue ninguna lógica en la búsqueda.

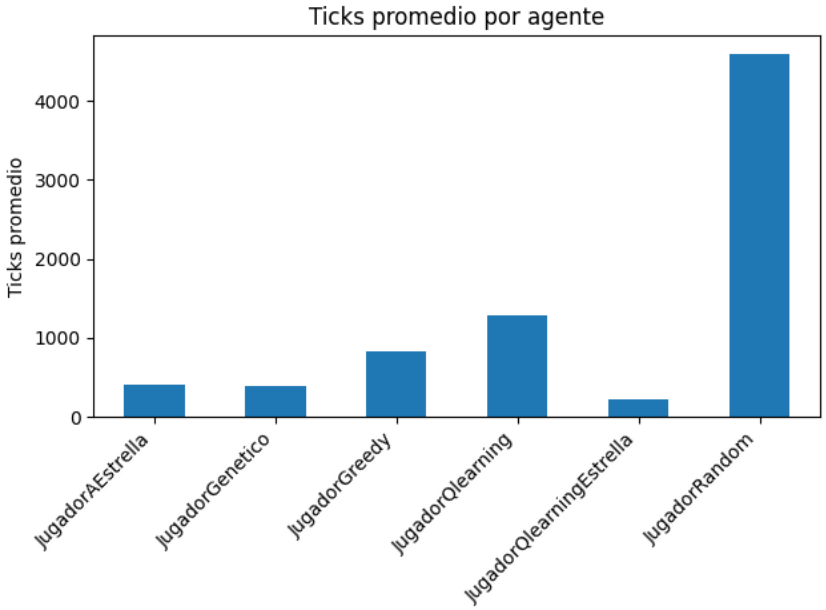

Figura 2: Convergencia de parámetros en jugadores genéticos

A pesar de lo observado en la gráfica 1, la gráfica 2 muestra que LRTA* y el algoritmo genético tardaron aproximadamente la misma cantidad de *ticks* (ciclos de cálculo y/o decisión). Resulta llamativo que el híbrido Q-learning + LRTA* sea el más eficiente en términos de **ticks**, requiriendo la menor cantidad para alcanzar la meta.

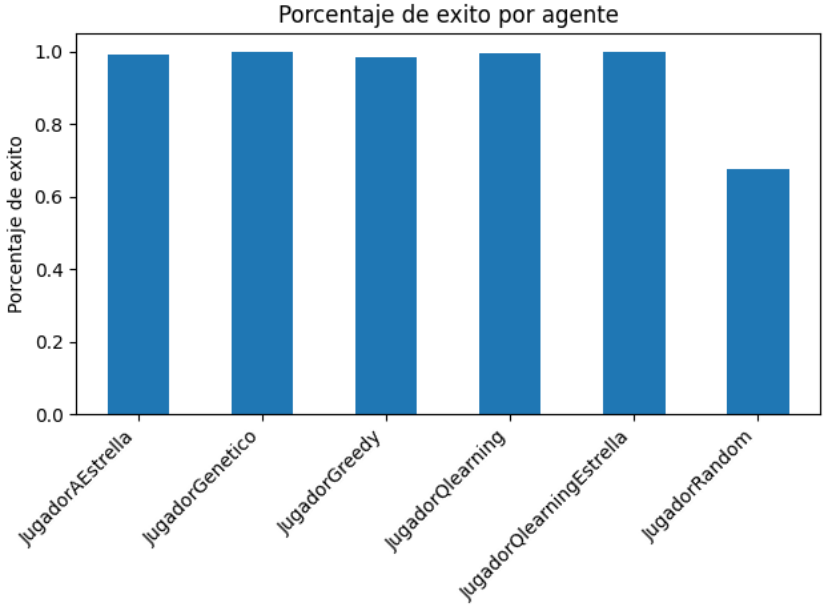

Figura 3: Porcentaje de éxito por agente

Como se aprecia en la gráfica 3, todos los algoritmos consiguieron alcanzar la meta, a excepción del algoritmo aleatorio, que alcanzó el límite de 10 000 ticks sin lograrlo.

3 RESULTADOS

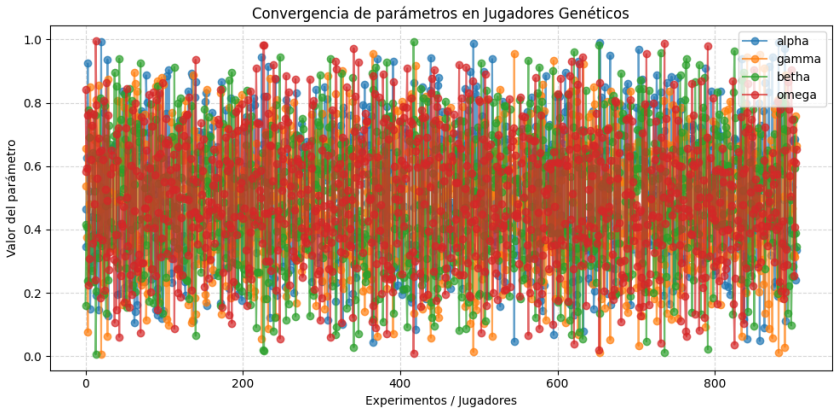

Figura 4: Convergencia de parámetros en Jugadores Genéticos

Como se observa en la gráfica 4, los parámetros no convergen en ningún momento, lo que indica que no se logra encontrar una combinación óptima entre Q-Learning y LRTA* para resolver este problema. Esto permite explicar por qué, en la gráfica 2, el híbrido Q-Learning + LRTA* requirió menos *ticks* que el algoritmo genético. Ambos algoritmos son esencialmente la misma base; sin embargo, el genético entrena y busca los mejores parámetros, mientras que el híbrido estándar utiliza por defecto una versión equilibrada (0.5 para todos los parámetros). Por lo tanto, la falta de convergencia del algoritmo genético puede justificar que el híbrido simple sea más eficiente en este caso.

4. Conclusiones y Reflexiones

A partir de nuestros experimentos, pudimos observar que el algoritmo LRTA* demuestra un rendimiento muy competitivo en comparación con los demás algoritmos evaluados. Sin embargo, el enfoque híbrido que combina Q-Learning con LRTA* se destacó como el más eficiente de todos, gracias a su capacidad de aprovechar tanto la información adquirida a partir de la experiencia como la información del estado actual. Por ello, consideramos que, siempre que sea posible y se cuenten con los recursos necesarios, los algoritmos híbridos ofrecen un desempeño superior en general.

Por otro lado, el algoritmo genético obtuvo resultados aceptables en términos de número de ticks, pero mostró un desempeño significativamente inferior en tiempo de ejecución, debido al costo de su fase de entrenamiento.

En conclusión, basándonos en los resultados obtenidos, recomendamos priorizar enfoques heurísticos e híbridos para la resolución de problemas dinámicos, ya que combinan eficiencia y efectividad de manera óptima.

5. Anexo

Repositorio Git del proyecto

https://github.com/4lehh/IA-Classic-vs-Genetic-Algorithm