Comparación de Algoritmos de Aprendizaje por Refuerzo en la Navegación de Robots Móviles: Q-Learning, Montecarlo y SARSA

David Fuentelsaz Rodríguez
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
davfuerod@alum.us.es

Nombre y apellidos alumno 2
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
Correos electrónicos UVUS y de contacto (si distinto)

***Resumen*—El objetivo principal de este trabajo es comparar tres algoritmos de aprendizaje por refuerzo: Q-Learning, Montecarlo y SARSA, en el contexto de la planificación de rutas para un robot móvil en un entorno con obstáculos. El estudio se centra en evaluar la eficiencia computacional, la convergencia y la robustez de cada algoritmo al navegar hacia un destino minimizando la posibilidad de colisión.**

Los resultados obtenidos muestran que cada algoritmo presenta ventajas y desventajas específicas en diferentes aspectos del problema planteado. Q-Learning demostró una rápida convergencia en la mayoría de los escenarios, mientras que Montecarlo ofreció una mejor exploración del espacio de estados. SARSA, por su parte, destacó en entornos altamente estocásticos debido a su enfoque de aprendizaje on-policy. Estas conclusiones ofrecen una guía para la selección del algoritmo más adecuado según las características del entorno y los requisitos del sistema.

***Palabras clave*—Inteligencia Artificial, Aprendizaje por Refuerzo, Q-Learning, Montecarlo, SARSA, Procesos de Decisión de Markov, Entornos Estocásticos, Política, Exploración y Explotación.**

I. Introducción

El aprendizaje por refuerzo es un tipo de aprendizaje automático que se enfoca en la toma de decisiones en entornos dinámicos y no deterministas. En este tipo de aprendizaje, el agente interactúa con el entorno y recibe recompensas o penalizaciones en función de sus acciones. El objetivo es maximizar las recompensas y minimizar las penalizaciones para encontrar la política óptima.

Esta técnica tiene una gran variedad de aplicaciones entre las que se encuentran predicciones financieras, robótica, videojuegos, medicina o cualquier problema de optimización [2] [3].

En el contexto de nuestro trabajo, nos centraremos en la aplicación del aprendizaje por refuerzo a la planificación de rutas para robots móviles. En este problema, un robot con ruedas debe encontrar una ruta segura y eficiente en un entorno con obstáculos. Aunque este problema puede parecer simple a primera vista, la presencia de obstáculos, la estocasticidad en el efecto de las acciones y la necesidad de optimizar la ruta para minimizar el tiempo y los recursos hacen que sea un desafío significativo.

Para evaluar la eficacia de los algoritmos de aprendizaje por refuerzo en este contexto, diseñamos tres mapas diferentes que varían en tamaño y porcentaje de obstáculos. Estos mapas nos permitirán comparar y analizar el rendimiento de los algoritmos en una variedad de escenarios. Comenzamos nuestro trabajo con el mapa de ejemplo proporcionado en la propuesta del trabajo, que tiene dimensiones de 15x51.

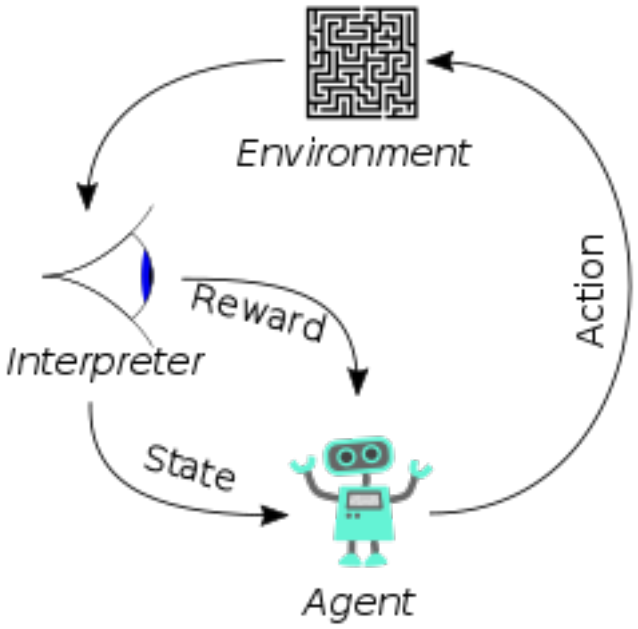

Fig. 1. Esquema del funcionamiento del aprendizaje por refuerzo.

II. Preliminares

En esta sección, proporcionamos una introducción a los conceptos fundamentales del aprendizaje por refuerzo y la planificación de rutas de robots móviles en entornos con obstáculos, así como una descripción de los algoritmos de aprendizaje por refuerzo que utilizaremos en nuestro trabajo.

A. Conceptos clave del aprendizaje por refuerzo

El aprendizaje por refuerzo (RL) es un tipo de aprendizaje automático que entrena al software para tomar decisiones y maximizar una recompensa en un entorno dado. Los componentes principales de un sistema de aprendizaje por refuerzo son:

- **Agente**: Es la parte del sistema que toma decisiones e interactúa con el entorno. En nuestro caso, el agente es el robot móvil que busca planificar rutas en un entorno con obstáculos.
- **Entorno**: Es el espacio en el que el agente se mueve y interactúa. En nuestro caso, el entorno es un mapa con obstáculos.
- **Acción**: Una acción es la decisión tomada por el agente en un estado particular. En nuestro caso, tenemos tenemos ocho acciones que representan cada uno de los posibles movimientos que puede efectuar el robot y una acción que representa que el agente permanezca en el mismo estado tras realizarla.
- **Recompensa**: La recompensa es la retroalimentación que el agente recibe del entorno después de tomar una acción en un estado dado. La recompensa puede ser positiva, neutra o negativa (esto sería una penalización).
- **Política**: La política es la estrategia utilizada por el agente para seleccionar acciones en función de los estados del entorno.
- **Exploración**: Es el proceso de probar nuevas acciones y recopilar información sobre el entorno. La exploración es esencial para que el agente descubra nuevas oportunidades y estrategias que podrían llevar a recompensas a largo plazo más altas.
- **Explotación**: Es el proceso de utilizar la información aprendida para tomar decisiones que maximicen la recompensa inmediata. La explotación es importante para que el agente obtenga la mayor recompensa posible en el momento presente.

B. Descripción de los algoritmos

A continuación, describimos los algoritmos de aprendizaje por refuerzo que utilizados en nuestro trabajo:

*1) **Q-Learning**:* El algoritmo Q-Learning es un método de aprendizaje por refuerzo basado en valores que se utiliza para encontrar la política óptima de selección de acciones en un entorno determinado. Este algoritmo es una variante del aprendizaje por refuerzo que utiliza una función de valor, denominada función Q, para determinar la mejor acción en cada estado del entorno. No requiere un modelo del entorno y puede manejar problemas con transiciones estocásticas y recompensas sin requerir adaptaciones [5].

El agente explora el entorno y actualiza la tabla Q utilizando la ecuación de Bellman modificada:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Donde:

- α es el factor de aprendizaje.
- $R(s,a)$ es la recompensa inmediata por tomar la acción a en el estado s.
- γ es el factor de descuento.
- s' es el estado siguiente al tomar la acción a en el estado s.
- $\max_{a'} Q(s',a')$ es el valor máximo esperado para cualquier acción posible en el estado s'.

Se trata de un algoritmo con un enfoque *off-policy*, lo que significa que aprende la función de valor óptima independientemente de la política que el agente esté siguiendo en el momento. Q-Learning actualiza la estimación de la función Q utilizando la recompensa inmediata y el valor máximo de la función Q para el siguiente estado y acción.

Hiperparámetros en Q-Learning

- **Tasa de aprendizaje (α):** Controla cuánto se actualiza la función Q con nuevas experiencias. Un valor alto de α permite un aprendizaje rápido pero puede causar oscilaciones, mientras que un valor bajo conduce a un aprendizaje más estable pero más lento.
- **Factor de descuento (γ):** Determina la importancia de las recompensas futuras. Un valor alto de γ hace que el agente valore más las recompensas futuras, lo que es esencial para tareas donde las decisiones tempranas afectan significativamente las recompensas futuras.
- **Estrategia de exploración (ϵ en ϵ-greedy):** Controla el balance entre exploración y explotación. Un valor alto de ϵ fomenta la exploración, lo que es crucial al inicio del aprendizaje, mientras que un valor bajo de ϵ fomenta la explotación de acciones conocidas.

*2) **Montecarlo**:*

*3) **SARSA**:* El algoritmo SARSA (State-Action-Reward-State-Action) es un algoritmo de aprendizaje por refuerzo que se utiliza para aprender una política óptima en un entorno de Markov discreto. Es bastante similar al algoritmo Q-Learning, aunque tiene una serie de diferencias. En primer lugar, SARSA es un algoritmo con un enfoque *on-policy*, lo que significa que el agente aprende una política óptima mientras sigue la misma política que está aprendiendo. Este algoritmo también utiliza una tabla Q(s, a) que almacena el valor esperado de tomar la acción a en el estado s. No obstante, la forma en la que se actualiza Q presenta una diferencia clave respecto a Q-Learning, como se puede comprobar en la siguiente expresión:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma Q(s',a') - Q(s,a)]$$

Donde:

- α es el factor de aprendizaje.
- $R(s,a)$ es la recompensa inmediata por tomar la acción a en el estado s.
- γ es el factor de descuento.

- s' es el estado siguiente al tomar la acción a en el estado s.
- a' es la acción tomada en el estado siguiente s'.

En este caso, se puede apreciar que la acción utilizada para actualizar la tabla Q en el siguiente estado es la seleccionada por la política actual. Al seguir un enfoque *on-policy*, SARSA, por lo general, suele ser un algoritmo más lento en la convergencia a la política óptima, especialmente en entornos complejos.

Hiperparámetros en SARSA

- **Tasa de aprendizaje (α):** Controla cuánto se actualiza la función Q con nuevas experiencias. Un valor alto de α permite un aprendizaje rápido pero puede causar oscilaciones, mientras que un valor bajo conduce a un aprendizaje más estable pero más lento.
- **Factor de descuento (γ):** Determina la importancia de las recompensas futuras. Un valor alto de γ hace que el agente valore más las recompensas futuras, lo que es esencial para tareas donde las decisiones tempranas afectan significativamente las recompensas futuras.
- **Estrategia de exploración (ϵ en ϵ-greedy):** Controla el balance entre exploración y explotación. Un valor alto de ϵ fomenta la exploración, lo que es crucial al inicio del aprendizaje, mientras que un valor bajo de ϵ fomenta la explotación de acciones conocidas.

C. Trabajos relacionados

Para un mejor entendimiento de cómo realizar la implementación de los algoritmos, se han revisado algunos trabajos realizados:

- Reinforcement Learning: An Introduction [1].

III. Metodología

En esta sección, describiremos el método implementado en nuestro trabajo. Abordaremos los siguientes aspectos:

- Descripción del entorno de trabajo.
- Detalles de la implementación de los algoritmos de aprendizaje por refuerzo (Q-Learning, Montecarlo y SARSA).
- Proceso experimental y evaluación de los resultados.

A. Descripción del entorno de trabajo

En el desarrollo de este trabajo, se empleó un entorno de simulación implementado en un Jupyter Notebook. El entorno de simulación se basó en un mapa de ejemplo proporcionado en el trabajo, el cual consiste en una cuadrícula que representa el entorno en el que se moverá el robot móvil. La cuadrícula está compuesta por casillas, algunas de las cuales contienen obstáculos (representadas con unos), mientras que otras están libres de ellos (representadas con ceros).

A partir del mapa de ejemplo, se generaron dos mapas adicionales para aumentar la variabilidad en los escenarios de prueba. Estos mapas proporcionaron diferentes configuraciones de entorno para evaluar el desempeño de los algoritmos de aprendizaje por refuerzo.

Las casillas iniciales y destino fueron seleccionadas aleatoriamente dentro de cada mapa generado, asegurándose de que ninguna de ellas fuera una casilla con obstáculos. Esta selección aleatoria garantizó la variabilidad en los escenarios de prueba y permitió evaluar el desempeño de los algoritmos en diferentes configuraciones del entorno.

En este entorno simulado, el robot móvil parte de una casilla inicial aleatoria y tiene como objetivo llegar a una casilla destino específica. Durante la simulación, el robot puede moverse entre casillas adyacentes en la cuadrícula, siguiendo una política determinada por los algoritmos de aprendizaje por refuerzo implementados. La interacción del robot con el entorno se realiza mediante acciones discretas.

Este enfoque de simulación proporcionó un entorno controlado y reproducible para evaluar el rendimiento de los algoritmos de aprendizaje por refuerzo en la planificación de rutas de robots móviles. Además, permitió explorar diferentes estrategias de navegación y analizar su efectividad en la resolución de problemas de planificación de rutas en entornos con obstáculos.

```
16 3
1111111111111111111
1000000010000000001
1000000010000000001
1000000000001000001
1000000000001000001
1111111111111111111
```

Fig. 2. Ejemplo mapa generado con la casilla objetivo.

B. Implementación de los algoritmos de aprendizaje por refuerzo

Funciones auxiliares

Aquí se describen las funciones auxiliares que han sido utilizadas en la implementación de los algoritmos de aprendizaje por refuerzo.

1. Método para leer el mapa de un fichero de texto

lee_mapa(*fichero*)
Entrada: Nombre del archivo *fichero*
Salida: Matriz que representa el mapa y dimensiones del mapa
Algoritmo:
1 **Leer las líneas del archivo *fichero* y convertirlas a matriz**
2 **Devolver la matriz y sus dimensiones**

Fig. 3. Pseudocódigo de la función `lee_mapa`.

2. Método para comprobar si una casilla contiene un obstáculos

hay_colision(*estado*)
Entrada: Casilla en la que se encuentra el robot *estado*
Salida: Valor booleano que indica si hay colisión
Algoritmo:
1 **Verificar si hay un obstáculo en *estado***
2 **Devolver true si hay colisión, false en caso contrario**

Fig. 4. Pseudocódigo del método `hay_colision`.

3. Método para aplicar una acción en un estado

aplica_accion(*estado*, *accion*)
Entrada: *estado* y *accion*
Salida: Nuevo *estado*
Algoritmo:
1 **Si hay colisión entonces**
2 **Devolver *estado***
3 $x, y \leftarrow$ **Coordenadas de *estado***
4 **Si *accion* es norte entonces**
5 $y \leftarrow y + 1$
6 **Si no, si *accion* es sur entonces**
7 $y \leftarrow y - 1$
8 **Si no, si *accion* es este entonces**
9 $x \leftarrow x + 1$
10 **Si no, si *accion* es oeste entonces**
11 $x \leftarrow x - 1$
12 **Sino, si *accion* es noreste entonces**
13 $x, y \leftarrow x + 1, y + 1$
14 **Si no, si *accion* es sureste entonces**
15 $x, y \leftarrow x + 1, y - 1$
16 **Si no, si *accion* es suroeste entonces**
17 $x, y \leftarrow x - 1, y - 1$
18 **Si no, si *accion* es noroeste entonces**
19 $x, y \leftarrow x - 1, y + 1$
20 **Devolver** (x, y) **como nuevo *estado***

Fig. 5. Pseudocódigo de `aplica_accion`.

4. Método para obtener los estados libres de obstáculos

estados_sin_obstaculos()
Salida: Lista de estados sin obstáculos
Algoritmo:
1 **Inicializar una lista vacía llamada *estados_sin_obstaculos***
2 *Para cada* **estado *en nav_estados*** *hacer*
3 *Si no hay_colision*$(estado)$ *entonces*
4 **Agregar el estado a la lista *estados_sin_obstaculos***
5 **Devolver la lista *estados_sin_obstaculos***

Fig. 6. Pseudocódigo detallado de `estados_sin_obstaculos`.

Siendo *nav_estados* la variable global con todos los estados del entorno.

5. Método para obtener los posibles desvíos del agente al realizar una acción

obtiene_posibles_errores(*accion*)
Entrada: *accion*
Salida: Lista de posibles acciones erróneas
Algoritmo:
1 **Si *accion* es norte entonces**
2 ***errores*** $\leftarrow$ **['NE', 'NO']**
3 **Si no, si *accion* es sur entonces**
4 ***errores*** $\leftarrow$ **['SE', 'SO']**
5 **Si no, si *accion* es este entonces**
6 ***errores*** $\leftarrow$ **['NE', 'SE']**
7 **Si no, si *accion* es oeste entonces**
8 ***errores*** $\leftarrow$ **['NO', 'SO']**
9 **Si no, si *accion* es noreste entonces**
10 ***errores*** $\leftarrow$ **['N', 'E']**
11 **Si no, si *accion* es noroeste entonces**
12 ***errores*** $\leftarrow$ **['N', 'O']**
13 **Si no, si *accion* es sureste entonces**
14 ***errores*** $\leftarrow$ **['S', 'E']**
15 **Si no, si *accion* es suroeste entonces**
16 ***errores*** $\leftarrow$ **['S', 'O']**
17 **Si no**
18 ***errores*** $\leftarrow$ **[]**
19 **Devolver *errores***

Fig. 7. Pseudocódigo de `obtiene_posibles_errores`.

6. Método para escoger una acción a partir de la política $\epsilon - greedy$

escoger_accion(*estado*, *epsilon*)
Entrada: Estado *estado* y valor de epsilon *epsilon*
Salida: Acción seleccionada
Algoritmo:
1 **Si *estado* no está en la tabla Q entonces**
2 **Inicializar los valores de *estado* a cero en la tabla Q**
3
4 **Si un número aleatorio entre 0 y 1 es menor que *epsilon***
5 **Seleccionar una acción aleatoria de entre todas las posibles**
6 **Si no**
7 **Seleccionar la mejor acción conocida**
8
9 **Devolver la acción seleccionada**

Fig. 8. Pseudocódigo de `escoger_accion`.

7. Método para obtener la política a partir de los valores de la tabla Q

obtener_politica(Q)
Entrada: Tabla Q
Salida: Política basada en Q
Algoritmo:

```
Inicializar politica como diccionario vacío
Para cada estado en nav_estados hacer
  Si estado está en Q_table entonces
    politica[estado] ← índice de max(Q_table[estado])
  Sino
    politica[estado] ← 0
Retornar politica
```

Fig. 9. Pseudocódigo de la función `obtener_politica`.

8. Método para obtener la recompensa de aplicar una acción a un estado

obtiene_recompensa($estado, accion$)
Entrada: Estado actual *estado*, acción a tomar *accion*
Salida: Recompensa correspondiente
Algoritmo:

```
x, y ← estado
Si estado es igual a destino entonces
  devolver recompensa_objetivo_alcanzado
Si hay colisión en estado entonces
  devolver -penalizacion_colision
Si accion es igual a 'esperar' entonces
  devolver -penalizacion_esperar
Para cada accion_error
en obtiene_posibles_errores(accion) hacer
  estado_vecino ← aplica_accion(estado, accion_error)
  Si hay colisión en estado_vecino entonces
    devolver -penalizacion_casilla_adyacente_obstaculo
```

Fig. 10. Pseudocódigo de la función `obtiene_recompensa`.

En este caso, *recompensa_objetivo_alcanzado*, *penalizacion_colision*, *penalizacion_esperar* y *penalizacion_casilla_adyacente_obstaculo* son variables globales que se usan en el método para definir el valor de la recompensa en diferentes situaciones.

Tras presentar todas las funciones auxiliares utilizadas en la implementación de los algoritmos, procedemos a detallar la implementación de Q-Learning, Montecarlo y SARSA.

Q-Learning

En primer lugar nos encontramos con el algoritmo de Q-Learning.

```
Q-Learning:
Inicializar Q como diccionario vacío
Parámetros: α = 0.2, γ = 0.9, ε = 0.3,
epocas = 5000, max_pasos = 100
Para cada época:
  Estado aleatorio sin obstáculos
  Para cada paso:
  accion_index ← escoger_accion(estado, ε)
  accion ← nav_acciones[accion_index]
  Aplicar acción, obtener nuevo estado y recompensa
Si estado no está en Q_table entonces
    Inicializar Q_table[estado] con ceros
Fin Si
Si nuevo_estado no está en Q_table entonces
    Inicializar Q_table[nuevo_estado] con ceros
Fin Si
mejor_accion_nueva ← máximo de Q_table[nuevo_estado]
Q_table[estado][accion] ← Q_table[estado][accion] +
 alpha * (recompensa + gamma * mejor_accion_nueva -
 Q_table[estado][accion])
  Si estado destino o terminal, terminar
```

Fig. 11. Pseudocódigo del algoritmo Q-Learning.

REFERENCIAS

[1] Richard S. Sutton y Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, 2018.
[2] https://www.codificandobits.com/curso/aprendizaje-por-refuerzo-nivel-basico/2-ejemplos-reales-aplicacion-aprendizaje-por-refuerzo/
[3] https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/
[4] K. Elissa, "Title of paper if known," unpublished.
[5] https://es.wikipedia.org/wiki/Q-learning
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.