Planificación de Transporte en Aeropuertos con Aprendizaje por Refuerzo

Sergio Madrid Pérez, Jorge Haces Fuertes

Índice

1. Introducción

El problema de planificación en aeropuertos es fundamental para garantizar la eficiencia del transporte de equipajes. Tradicionalmente, se han utilizado algoritmos de planificación clásicos, pero en este trabajo exploramos el uso de aprendizaje por refuerzo (RL) para optimizar las rutas de transporte.

Nuestro objetivo es aplicar el algoritmo Q-learning para mejorar la toma de decisiones en este dominio. Utilizamos PDDLGym, una librería basada en PDDL (Planning Domain Definition Language), para modelar el entorno y evaluar el rendimiento del agente.

2. Adaptación a PDDLGym

Para poder emplear nuestro dominio y problema empleado en la práctica de PDDL, fue necesario adaptar el código teniendo en cuenta los requisitos de PDDLGym, así como diseñar un problema abordable por el algoritmo Q-learning. Para ello, se hicieron las siguientes modificaciones:

- Se eliminaron las definiciones con la expresión `either`. El siguiente predicado at:

  ```
  (at ?x - (either maquina vagon equipaje) ?y - posicion)
  ```

  se eliminó y se añadió un predicado distinto para cada tipo:

  ```
  (at_maquina ?x - maquina ?y - posicion)
  (at_vagon ?x - vagon ?y - posicion)
  (at_equipaje ?x - equipaje ?y - posicion)
  ```

- Se redujo el tamaño del problema para hacerlo más abordable por el agente de RL:

  - Las puertas 3, 4, 7 y 8 se eliminaron.
  - La capacidad de los vagones se redujo a un solo equipaje.
  - Se definió un solo objetivo: transportar un equipaje desde su punto de inicio a su destino. El equipaje puede ser sospechoso o no sospechoso.

  El problema original se muestra en la Figura 1, mientras que la representación del problema reducido tras aplicar las modificaciones anteriores se puede observar en la Figura 2.

- Eliminamos el conteo, ya que es innecesario cuando la capacidad de los vagones es 1. Se eliminaron los siguientes predicados:

  `(carga-actual ?v - vagon ?n - numero)`
  `(next ?v - vagon ?n1 - numero ?n2 - numero)`

  y se cambió el predicado `vacio` para indicar que un vagón no contiene ningún equipaje:

  `(vacio ?n - numero)` $\rightarrow$ `(vacio ?v - vagon)`

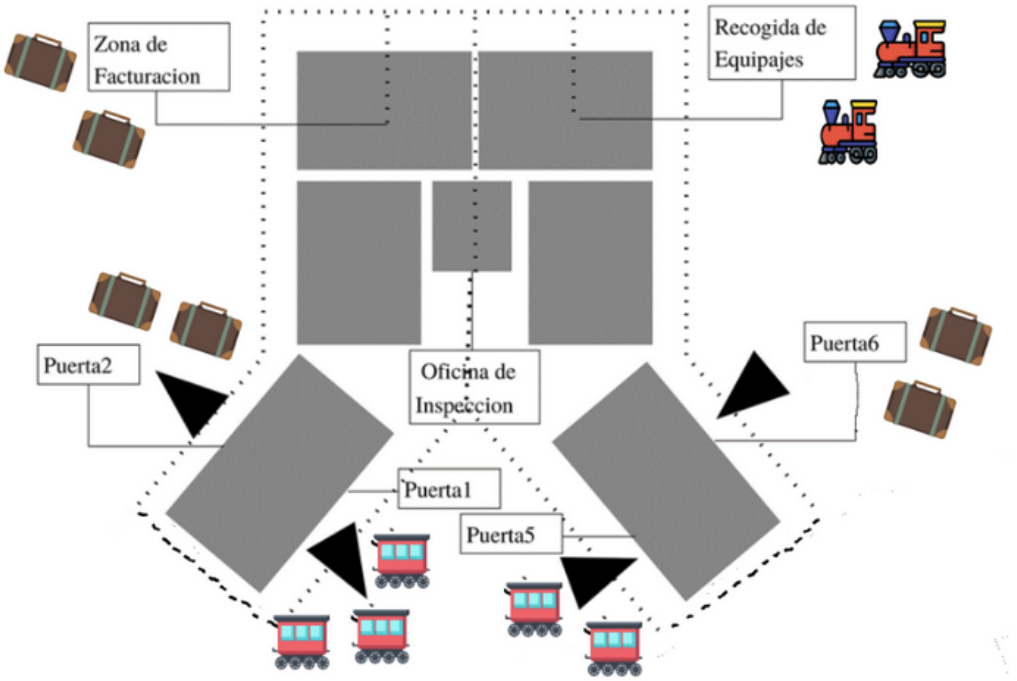

Figura 1: Representación del problema original.

3. Implementación algoritmo Q-learning

En esta sección se describe la implementación del algoritmo Q-learning, un enfoque de aprendizaje por refuerzo utilizado para que un agente aprenda a tomar decisiones óptimas en un entorno mediante la maximización de una función de recompensa. A continuación se detallan los componentes clave de la implementación, comenzando por la estrategia de exploración y explotación, seguida de la estructura de la tabla Q utilizada para almacenar y actualizar el conocimiento del agente.

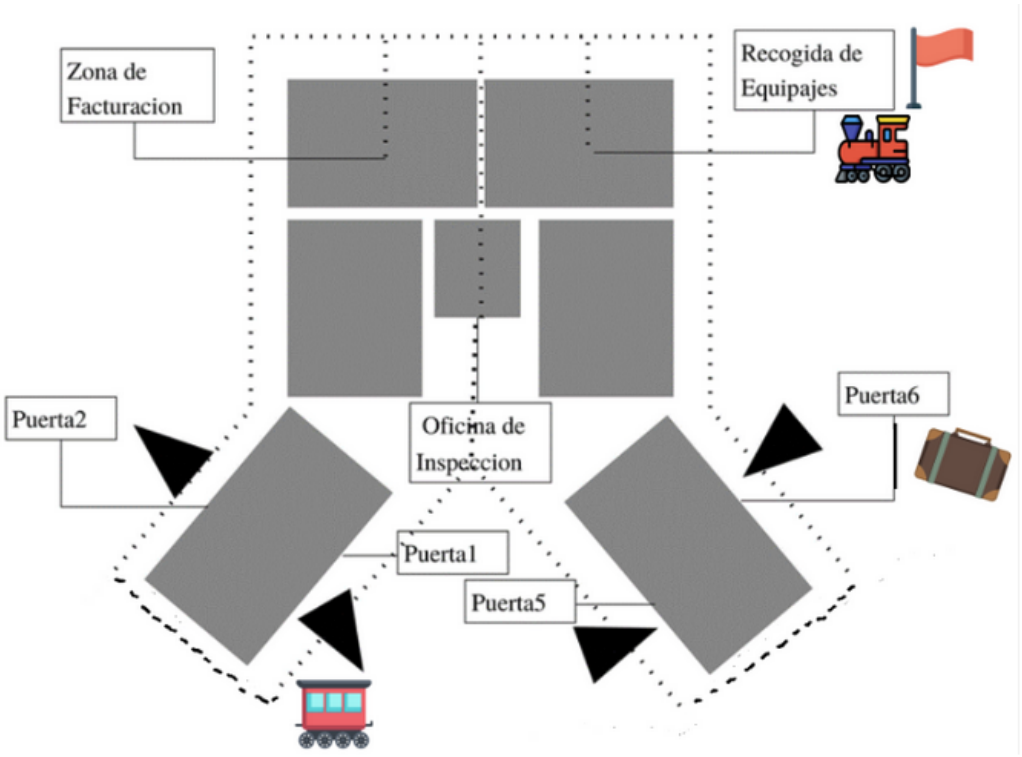

Figura 2: Representación del problema reducido para PDDLGym

3.1. Exploración y explotación

El algoritmo Q-learning emplea una estrategia de exploración y explotación para balancear el aprendizaje del agente. Durante el proceso de exploración, el agente toma acciones aleatorias, lo que le permite aprender más sobre el entorno y descubrir acciones que puedan ser útiles, incluso si no las ha probado antes. Por otro lado, en la fase de explotación, el agente toma las mejores decisiones basándose en el conocimiento acumulado en la tabla Q, seleccionando las acciones con el mayor valor esperado.

Como método de exploración se empleó una política ϵ-greedy, la cual consiste en explorar con una probabilidad ϵ y explotar con probabilidad $1-\epsilon$. En cuanto a la actualización del umbral ϵ, se implementó una estrategia de decaimiento exponencial, es decir, ϵ comienza con un valor inicial y en cada iteración del algoritmo se multiplica su valor actual por una tasa de decrecimiento λ hasta llegar al valor mínimo establecido. La Figura 3 muestra la evolución del valor de ϵ a lo largo de los episodios para un valor inicial de 1 y λ igual a 0.99.

$$\epsilon_{t+1} = \max(\epsilon_t \cdot \lambda, \epsilon_{min})$$

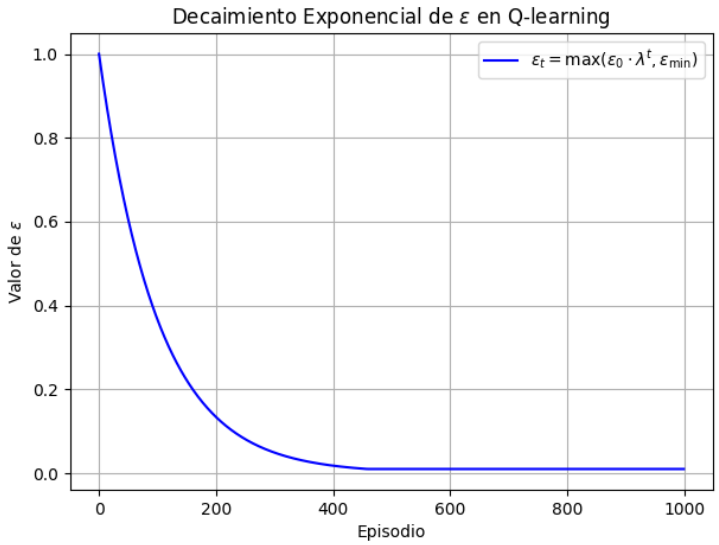

Figura 3: Decrecimiento de ϵ para $\epsilon_0 = 1$ y $\lambda = 0{,}99$.

3.2. Tabla Q

La tabla Q es la estructura clave que almacena el conocimiento aprendido por el agente a lo largo del tiempo. Para su implementación se optó por usar un diccionario en el que las claves serán los estados y los valores serán un diccionario cuyas claves serán las acciones y los valores serán el valor de la tabla Q para un determinado estado y una acción concreta.

Con esta definición de la tabla Q logramos que cada nuevo estado se añada dinámicamente. Además, cada estado solo contendrá las acciones que son válidas para dicho estado. De esta forma, nos aseguramos de que en la fase de explotación no se ejecuten acciones inválidas. El código empleado se muestra en la Figura 4.

```
def ensure_state_in_qtable(state):
    if state not in Q_table:
        Q_table[state] = {}
    valid_actions = env.action_space.all_ground_literals(state)
    for a in valid_actions:
        if a not in Q_table[state]:
            Q_table[state][a] = 0.0
```

Figura 4: Definición de la tabla Q mediante un diccionario. Para cada estado solo se añaden las acciones válidas.

La actualización de los valores en la tabla Q se realiza utilizando la ecuación de actualización del algoritmo Q-learning:

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

Donde:

- $Q(S_t, A_t)$ es el valor actual de la acción A en el estado S.
- α es la tasa de aprendizaje (learning rate), que controla cuánto se actualiza el valor.
- R_{t+1} es la recompensa obtenida tras tomar la acción A en el estado S.
- γ es el factor de descuento, que pondera la importancia de las recompensas futuras.
- $\max_a Q(S_{t+1}, a)$ representa el valor máximo esperado de las acciones en el nuevo estado S_{t+1}.

Este proceso permite que el agente aprenda gradualmente los valores óptimos de Q mediante la interacción con el entorno, favoreciendo la selección de acciones que maximicen la recompensa acumulada a largo plazo.

4. Experimentos

Para evaluar el rendimiento del algoritmo Q-learning en nuestro problema de transporte de equipajes en un aeropuerto, realizamos una serie de experimentos variando diferentes hiperparámetros clave.

Cada experimento se realizó con 4 semillas distintas para aumentar la robustez de los resultados. Se mide la convergencia del aprendizaje a través del número de pasos necesarios para completar una planificación, promediando los resultados obtenidos en distintas ejecuciones. En cada experimento, el resto de parámetros fuera de la comparación se establecieron a sus valores predeterminados: $\epsilon_0 = 1{,}0$, $\lambda = 0{,}99$, $\alpha = 9{,}2$, $\gamma = 0{,}99$. El número máximo de episodios se fijó a 50, mientras que el número máximo de iteraciones en cada episodio es de 2000.

4.1. Epsilon inicial

El valor inicial de ϵ (ϵ_0) controla la cantidad de exploración en las primeras etapas del entrenamiento. Un valor muy alto puede llevar a demasiada

exploración y aprendizaje ineficiente, mientras que un valor bajo puede llevar a una rápida explotación de políticas subóptimas.

Para evaluar el impacto de ϵ_0, se probaron tres valores diferentes: 0.8, 0.9 y 1. Los resultados mostrados en la Figura 5 indican que el rendimiento ha sido similar en todos los casos. Además, se comprobó que con valores inferiores a 0.8 el algoritmo tiene dificultades para converger, ya que el valor de ϵ llega muy rápido al mínimo y la exploración es insuficiente.

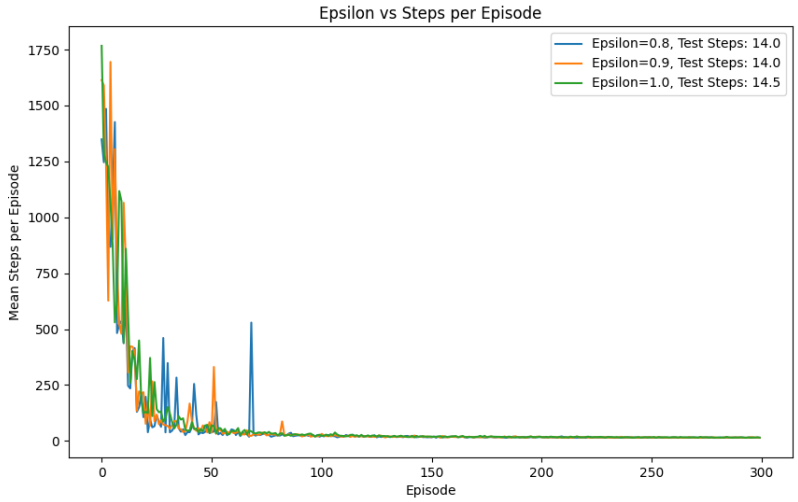

Figura 5: Media de pasos en cada episodio para cada valor inicial de epsilon ϵ_0.

4.2. Tasa de decaimiento

La tasa de decaimiento de ϵ (λ) es crucial para la transición entre exploración y explotación. Un ϵ que decae rápidamente puede llevar a una convergencia prematura en una política subóptima, mientras que un ϵ con decaimiento lento prolonga la exploración innecesariamente.

Se llevó a cabo una comparación de varios valores de λ (0.995, 0.99 y 0.9). La Figura 6 muestra que los mejores resultados se obtuvieron con un valor de 0.99. Además, la línea horizontal en el caso de 0.9, indica que en una prueba el algoritmo no encontró ninguna solución, llegando a máximos de iteraciones en cada episodio. Esto demuestra que valores cercanos o inferiores a 0.9, presentan problemas para converger.

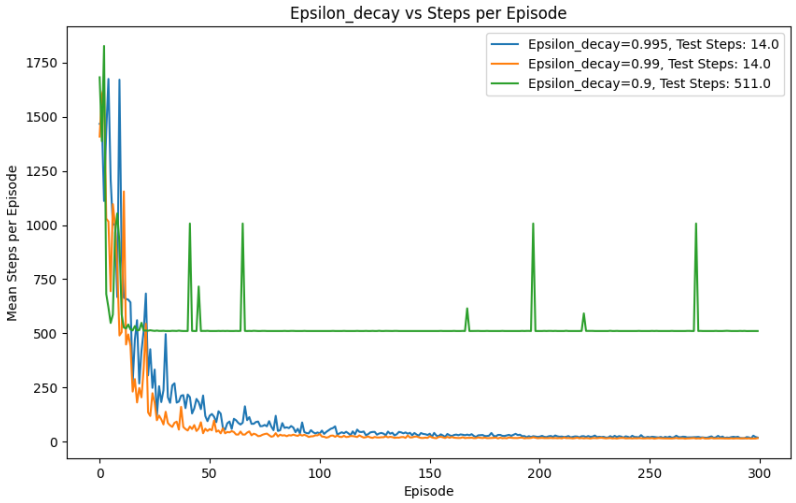

Figura 6: Media de pasos en cada episodio para cada valor de la tasa de decaimiento λ.

4.3. Learning rate

El learning rate α determina qué tan rápido se actualizan los valores de la tabla Q. Un valor demasiado alto puede provocar oscilaciones en el aprendizaje, mientras que un valor muy bajo puede hacer que el agente aprenda lentamente.

En la Figura 7 se muestra el número de pasos por episodio para diferentes valores de α (0.1, 0.2 y 0.5). En esto caso, se observa que los resultados obtenidos son prácticamente idénticos para cada valor probado, aunque parece que los valores superiores disminuyen ligeramente el tiempo necesario para converger.

4.4. Factor de descuento

El factor de descuento γ es un parámetro que define la importancia de las recompensas futuras. Valores cercanos a 1 favorecen estrategias a largo plazo, mientras que valores bajos priorizan recompensas inmediatas.

Para evaluar el impacto de γ, se probaron tres valores diferentes: 0.99, 0.95 y 0.9. A la vista de los resultados mostrados en la Figura 8, podemos concluir que no se observan diferencias significativas en el rendimiento al variar el valor de gamma con los valores probados. Sin embargo, se comprobó experimentalmente que valores inferiores de γ impiden que el algoritmo converja correctamente.

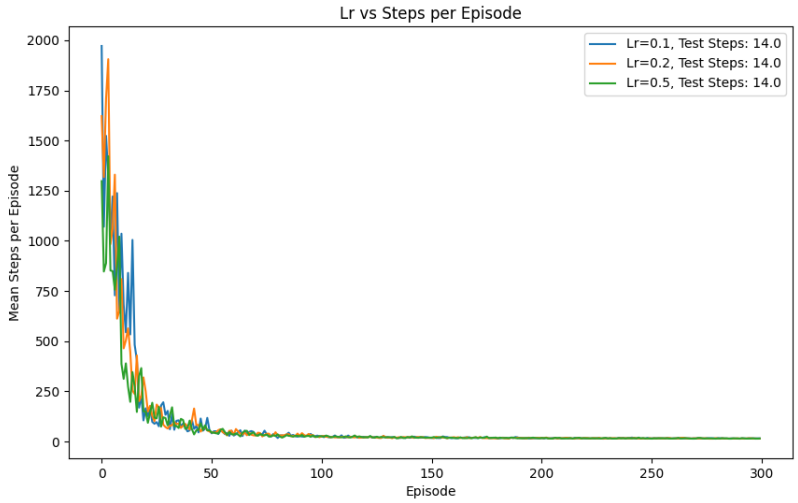

Figura 7: Media de pasos en cada episodio para cada valor de learning rate α.

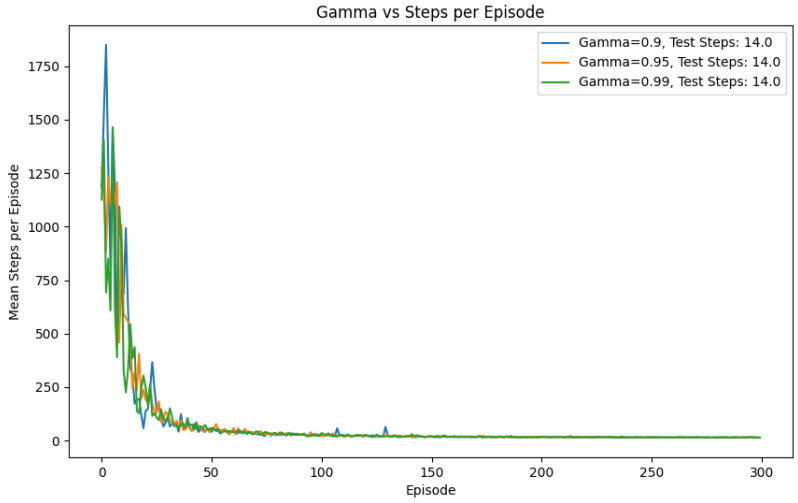

Figura 8: Media de pasos en cada episodio para cada valor del factor de descuento γ.

4.5. Tamaño del problema

Como último experimento se probó a aumentar el tamaño del problema, para ver hasta cómo de escalable es el problema al resolverlo mediante el

algoritmo Q-learning. Tras varias pruebas, se comprobó que añadir un nuevo vagón o una máquina no era viable, ya que el algoritmo no lograba encontrar una solución al problema. Esto es debido a dos motivos: los estados contienen una gran cantidad de literales y los vagones pueden desengancharse de la máquina en cualquier momento. Por estas dos razones, es muy complicado que los estados se repitan suficientemente como para aprender los valores de la tabla Q y encontrar un camino.

A continuación, se probó a aumentar el número de equipajes y se comprobó que el número límite de equipajes, a partir del cual el algoritmo no encontraba solución, es de 2 equipajes. Por tanto, se diseñaron varios problemas, mostrados en la Tabla 1 con distinto número de equipajes sospechosos y no sospechosos. Para cada uno de estos problemas, se obtuvo el mejor plan encontrado por el algoritmo Q-learning, así como los planes calculados por los planificadores empleados en la primera parte de la práctica. Además, se calculó el tiempo de ejecución en cada caso para hacer una comparación de los resultados.

Problema	**Máq.**	**Vag.**	**Equip.**	
			Sosp.	**No sosp.**
1	1	1	0	1
2	1	1	1	0
3	1	1	1	1
4	1	1	2	0

Tabla 1: Descripción de los problemas con el número de máquinas, vagones y equipajes.

Los resultados obtenidos por el algoritmo Q-learning se presentan en la Tabla 2, mientras que los resultados obtenidos por cada planificador (MetricFF, LPG, y Optic) se recogen en la Tabla 3. Al comparar los tiempos de ejecución, se observa que los planificadores muestran tiempos significativamente menores en comparación con el algoritmo Q-learning en todos los problemas. En particular, los planificadores terminan en tiempos cercanos a 0.02s para la mayoría de los casos, mientras que el algoritmo Q-learning tarda entre 7.30s y 57.61s, dependiendo de la complejidad del problema.

En cuanto al número de acciones, se puede observar como, en algunas ocasiones, el algoritmo Q-learning encuentra planes ligeramente mejores que los obtenidos con planificadores LPG y Optic. Sin embargo, el planificador MetricFF ha conseguido planes iguales o mejores que los obtenidos con Q-learning en todos los casos.

Problema	RL	
	Acciones	**Tiempo**
1	9	7.30s
2	14	8.55s
3	20	23.19s
4	25	57.61s

Tabla 2: Resultados obtenidos para Q-learning en cada problema.

Problema	MetricFF		LPG		Optic	
	Acciones	**Tiempo**	**Acciones**	**Tiempo**	**Acciones**	**Tiempo**
1	9	0.00s	9	0.02s	9	0.02s
2	14	0.00s	17	0.02s	17	0.02s
3	18	0.00s	21	0.02s	21	0.02s
4	25	0.00s	25	0.02s	25	0.02s

Tabla 3: Resultados obtenidos en los distintos problemas con los planificadores.

5. Conclusiones

En este trabajo se ha explorado el uso del algoritmo de Q-learning para la planificación de transporte de equipajes en un aeropuerto, con el objetivo de optimizar la asignación de máquinas, vagones y equipajes de manera eficiente. Tras implementar y probar el algoritmo en un dominio adaptado para PDDLGym, se obtuvieron diversos resultados que permiten realizar varias conclusiones.

El algoritmo Q-learning ha demostrado ser capaz de encontrar soluciones a los problemas de planificación, pero su eficiencia decrece a medida que aumenta el tamaño del problema, especialmente cuando se añaden más máquinas o vagones. Esto se debe a la gran cantidad de estados posibles, lo que provoca una falta de repetición de estados suficientes para que el agente aprenda de manera efectiva. A pesar de esto, los resultados muestran que el algoritmo es útil para problemas más pequeños y de menor complejidad.

Al comparar los resultados del algoritmo Q-learning con los de los planificadores tradicionales (MetricFF, LPG y Optic), se observa que estos últimos son considerablemente más rápidos, resolviendo los problemas en tiempos cercanos a 0.02 segundos. Sin embargo, en algunos casos, el algoritmo Q-learning consigue planes con un número de acciones ligeramente superior, aunque los mejores resultados los ha obtenido MetricFF.

En conclusión, el uso de Q-learning para la planificación de transporte

de equipajes en aeropuertos es una opción válida, especialmente para problemas de tamaño pequeño, pero los planificadores tradicionales siguen siendo más adecuados para entornos más complejos debido a su mayor eficiencia en términos de tiempo de ejecución y número de acciones.