Razonamiento y planificación automática

Resolución de problema mediante búsqueda heurística

Actividad 2

Pérez Fernández, Iván
ivan.perez540@comunidadunir.net

Acaso Nerín, Miguel
miguel.ascaso060@comunidadunir.net

Lecea Salinas, Javier
javier.lecea792@comunidadunir.net

Zarranz Calvo, Jesús Miguel
jesusmiguel.zarranz705@comunidadunir.net

1 de febrero de 2021

1. Desarrollo de la actividad

1.1. Análisis

La actividad propone resolver el problema de traslado de inventarios desde una posición A a una posición B. Estos inventarios solamente puede ser movidos mediante un robot que los cargará y transportará.

Este robot, está situado dentro del plano de actuación (al igual que los tres inventarios) y debe tener en cuenta que hay dos casillas que simulan paredes, de modo que el robot no podrá atravesarlas.

El estado inicial del problema es el siguiente:

	0	1	2	3
0	M1	#		M3
1		#		
2	M2		R	
3				

Figura 1: Estado inicial

donde:

- M1 es el inventario 1 situado en [0,0]
- M2 es el inventario 2 situado en [2,0]
- M3 es el inventario 3 situado en [0,3]
- R es el robot situado en [2,2]
- # son las paredes situadas en [0,1] y [1,1]

Partiendo de este estado, el robot debe ser capaz de mover los inventarios a determinadas posiciones destino, de modo que el estado objetivo debe quedar como en la siguiente imagen:

	0	1	2	3
0		#		
1		#		
2				
3		M3	M2	M1

Figura 2: Estado objetivo

Antes de ejecutar el plan de acción, debemos tener en cuenta una serie de restricciones, como son las paredes y los propios inventarios. Es decir, el robot no podrá atravesar ninguna de las paredes definidas ni los inventarios que se encuentre en la ruta. Esto lo explicaremos en la siguiente sección con más detalle.

1.2. Pruebas realizadas

El robot debe hacer uso del algoritmo A* para encontrar la ruta entre dos posiciones. Además, debemos tener en cuenta de que el robot solamente puede moverse hacia arriba, abajo, derecha o izquierda, por lo tanto no podrá acceder a las celdas vecinas en diagonal. Por ello se nos indica utilizar la distancia de Manhattan entre dos puntos, que viene definida por la siguiente función:

$$dist(x_i, x_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| \tag{1}$$

Con esto en mente, el primer problema que se puede observar es, partiendo del estado inicial, el robot sería incapaz de llegar al inventario M1, ya que llegaría a alguna celda para la cual no encuentre una celda vecina 'visitable', ya sea porque es una pared o porque ahí se encuentra el inventario M2:

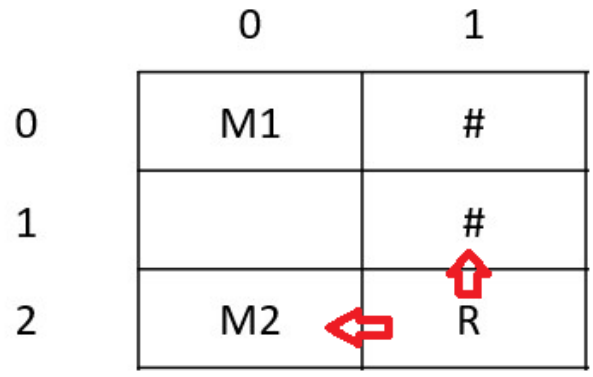

Figura 3: Robot incapaz de llegar a M1

Teniendo en cuenta esto, podemos considerar dos opciones para realizar la tarea (las cuales han sido implementadas en código fuente del notebook de Jupyter):

1. Indicarle al robot el orden en el que debe mover los inventarios

   Este sería el caso más sencillo de implementar. Lo primero sería mover M2 a su posición destino, de ese modo el robot ya tendría el camino despejado y sería capaz de encontrar rutas tanto para llegar a M1 como M3. Los movimientos a realizar serían:

   a) Robot va desde posición inicial hasta M2 y lo carga

   b) Robot mueve M2 desde posición origen a posición destino

   c) Robot va desde posición destino de M2 (donde se encuentra en ese instante) y llega hasta la posición de M1

d) Robot mueve M1 desde posición origen a posición destino

e) Robot va desde posición destino de M1 (donde se encuentra en ese instante) y llega hasta la posición de M3

f) Robot mueve M3 desde posición origen a posición destino

Una vez realizados estos movimientos, tendríamos todos los inventarios en su estado objetivo, por lo tanto el robot ha cumplido su cometido.

2. Implementar el algoritmo de tal forma que el robot sea capaz de mover los inventarios de forma autónoma sin independientemente del orden

   Como hemos descrito, la opción anterior funciona pero tiene sus limitaciones. Hemos querido darle un pequeño valor añadido a esta actividad y dotar al robot de una mayor *autonomía* a la hora de resolver el problema. En concreto, la única información que le damos al robot es la lista de tareas que tiene que realizar, es decir, mueve M1 de punto A al punto B, mueve M2 del punto C al punto D... etc Con esto, independientemente del orden de las tareas y de la posición de los inventarios, el robot será capaz de resolver el problema. El pseudocódigo del algoritmo sería el siguiente:

```
tareasPendientes = [tarea1, tarea2, tarea3]
while tareasPendientes do
    for tarea in tareasPendientes do
        if encuentraRuta para tarea then
            mueveInventario
            eliminaTarea de tareasPendientes
        else
            continua
        end if
    end for
end while
```

   De este modo, si la tarea de mover el inventario M1 fuera la primera a realizar, detectaría que no es posible alcanzarlo ya que hay bloqueos en la ruta y procedería a realizar la siguiente tarea posible, iterando hasta que consiga realizar todas las tareas satisfactoriamente.

1.3. Plan de acción

En esta sección documentaremos los planes encontrados por el robot para realizar la tarea de mover los inventarios a las posiciones objetivo. Comentar que para mostrar las rutas de cada uno de los movimientos realizados por el robot, se ha implementado una función customizada en Python que permite dibujar la traza y ver la ejecución del plan de una forma visual.

Estos son los resultados obtenidos para el plan de acción:

1. Ruta para ir de R a M2: (2,2) → (2, 1) → (2, 0)

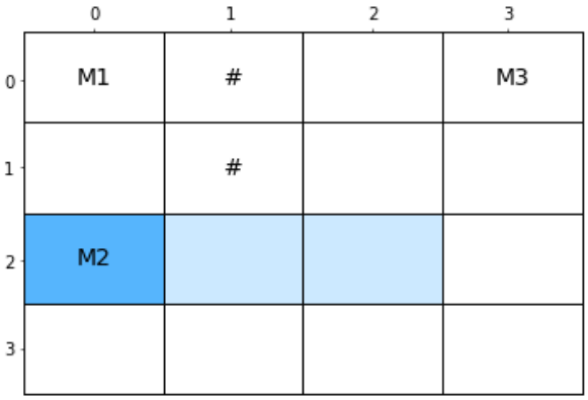

Figura 4: Robot carga el inventario M2

2. Ruta para mover M2 a posición destino: (2,0) → (2, 1) → (3, 1) → (3, 2)

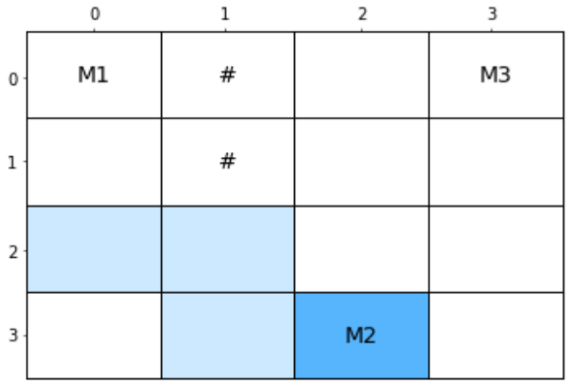

Figura 5: Robot mueve el inventario M2 a la posición destino

3. Ruta para ir de R a M1: (3,2) → (3, 1) → (2, 1) → (2, 0) → (1, 0) → (0, 0)

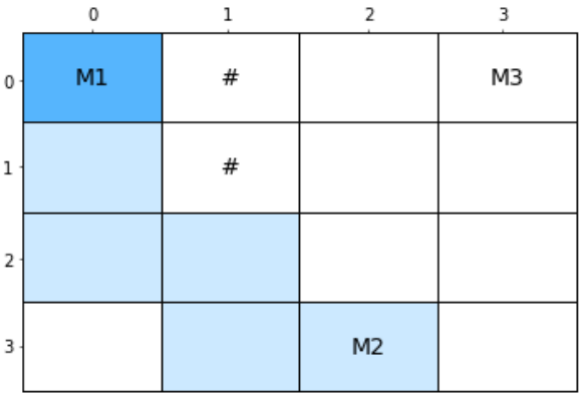

Figura 6: Robot carga el inventario M1

4. Ruta para mover M1 a posición destino: (0,0) → (1, 0) → (2, 0) → (2, 1) → (2, 2) → (2, 3) → (3, 3)

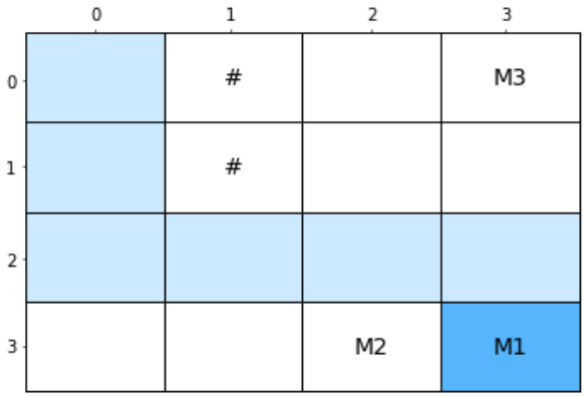

Figura 7: Robot mueve el inventario M1 a la posición destino

5. Ruta para ir de R a M3: (3,3) → (2, 3) → (1, 3) → (0, 3)

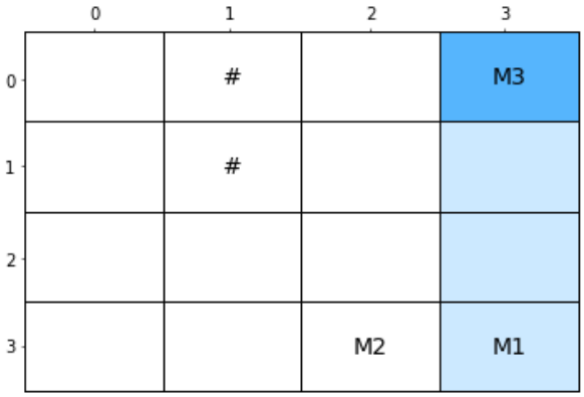

Figura 8: Robot carga el inventario M3

6. Ruta para mover M3 a posición destino: (0,3) → (1, 3) → (1, 2) → (2, 2) → (2, 1) → (3, 1)

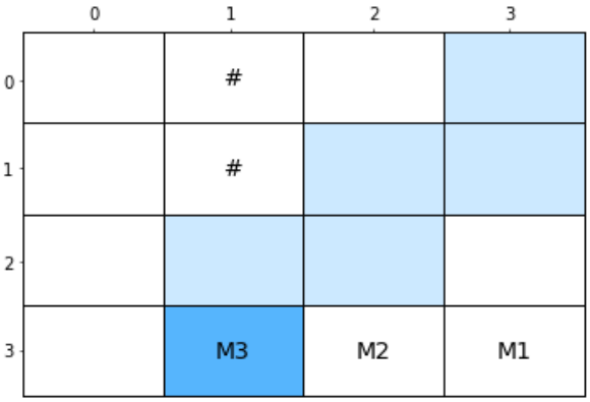

Figura 9: Robot mueve el inventario M3 a la posición destino

Estos son los resultados del plan realizado por el robot. Ahora, para corroborar la implementación del algoritmo es capaz de realizar las tareas incluso cuando para alguno de los inventarios no se encuentra ninguna ruta posible, se muestra la salida por consola del programa, donde se intenta como primera tarea mover M1:

```
Realizando tarea: mover M1
Buscando inventario...
Ninguna ruta fue encontrada... pasamos a la siguiente tarea
Realizando tarea: mover M2
Buscando inventario...
(2,2)-->(2,1)-->(2,0)
Inventario cargado!
(2,0)-->(2,1)-->(3,1)-->(3,2)
Inventario desplazado!
Tareas pendientes: 2
Realizando tarea: mover M1
Buscando inventario...
(3,2)-->(3,1)-->(2,1)-->(2,0)-->(1,0)-->(0,0)
Inventario cargado!
(0,0)-->(1,0)-->(2,0)-->(2,1)-->(2,2)-->(2,3)-->(3,3)
Inventario desplazado!
Tareas pendientes: 1
Realizando tarea: mover M3
Buscando inventario...
(3,3)-->(2,3)-->(1,3)-->(0,3)
Inventario cargado!
(0,3)-->(0,2)-->(1,2)-->(2,2)-->(2,1)-->(3,1)
Inventario desplazado!
Tareas pendientes: 0
```

Figura 10: Resultado de la automatización del proceso

Como se puede observar, el robot intenta ejecutar el plan de mover el inventario M1 primero, sin embargo (y como era de esperar) es capaz de encontrar ninguna ruta. Esto no obstante no supone ningún impedimento, puesto que continúa con la siguiente tarea y mueve M2, luego M3 y finalmente intenta encontrar una nueva ruta para M1, lo cual ahora sí es posible puesto que ya ha movido M2 y con ello desbloqueado el camino.

2. Dificultades encontradas

Cuando comenzamos a programar el algoritmo, detectamos algunos factores que no habíamos tenido en cuenta en un principio, por ejemplo:

- El cálculo de la posible posición del robot para cada nodo tiene en principio cuatro posibilidades (arriba, abajo, derecha e izquierda). Sin embargo, debemos tener en cuenta que si algunos de esos nodos vecinos son pared o inventario, no podemos tenerlos en cuenta. Además, teníamos que tener en cuenta también los límites del mapa, es decir, si el robot está en la posición (3,3) solamente podrá realizar dos movimientos, ya que se encuentra situado en una esquina. Todo esto está documentado con claridad en el notebook dentro del método `get_vecinos()`.
- La programación del algoritmo A* tampoco fue sencilla, ya que además queríamos que el código fuera entendible y no muy rebuscado, de forma cualquier persona con conocimientos básicos de programación

pueda entender cómo funciona. Para ello encontramos buenas referencias en la documentación de la asignatura y en investigaciones propias como el paper Ganapathy, 2016.

- Una vez tuvimos el problema resuelto en el notebook indicando al robot el orden en el que debía mover los inventarios, detectamos que en caso de que el M1 fuera el primero, nuestro código no funcionaba. El robot era incapaz de encontrar una ruta ya que se encontraba bloqueada por M2 y la pared, y el algoritmo entraba en bucle infinito. Tuvimos la idea de utilizar el manejo de excepciones en Python (véase Python-Software-Foundation, 2021) para controlar esta problemática, de modo que cuando el algoritmo A* no encuentra ninguna ruta, lanza una excepción. Esto nos ofrece la posibilidad de , mediante `try: y except:`, controlar si se ha encontrado una ruta posible o no y actuar en consecuencia.

Referencias

Ganapathy, V. (2016). Trajectory Planning of a Mobile Robot using Enhanced A-Star Algorithm. *Indian Journal of Science and Technology*, *7*(41), 1-10. https://doi.org/10.17485/ijst/2016/v9i41/93816

Python-Software-Foundation. (2021). Exception handling in Python. https://docs.python.org/3/tutorial/errors.html#handling-exceptions