TÉCNICAS DE LOS SISTEMAS INTELIGENTES.

Práctica 1. Desarrollo de agentes basado en técnicas de búsqueda heurística dentro del entorno GVGAI.

UNIVERSIDAD
DE GRANADA

Curso 2022/23

Jesús Miguel Rojas Gómez.
jesusjrg1400@correo.ugr.es

Contents

1 Resultados obtenidos

En la Figura 1 podemos ver los resultados obtenidos.

Algoritmo	Mapa	Runtime (acumulado)	Tamaño de la ruta	Nodos expandidos
Labyrinth (original)				
Dijkstra	6	1	36	89
	7	3	114	566
	8	5	808	2144
A*	6	1	36	84
	7	4	114	557
	8	8	808	2128
RTA*	6	0	40	40
	7	0	338	338
	8	0	3386	3386
LRTA*	6	0	60	60
	7	0	5658	5658
	8	TO	TO	TO
Labyrinth extended				
RTA*	6	0	85	85
	7	0	338	338
	8	0	3402	3402
LRTA*	6	0	151	151
	7	0	5658	5658
	8	TO	TO	TO

Figure 1: Resultados de los algoritmos ejecutados.

2 Descripción del agente de competición

Para la competición he utilizado el algoritmo A*, replanificando la ruta cada vez que el agente se encuentra con una casilla que cambia el mapa.

He decidido implementar este algoritmo ya que el problema que intentamos resolver tiene información completa, es decir, en todo momento, nuestro agente conoce donde se encuentran los muros, donde se encuentran las trampas y donde se encuentra la meta. Por lo tanto, no tiene sentido utilizar un algoritmo de búsqueda en online, ya que incrementaría el costo computacional sin necesidad alguna. Por otro lado, utilizo el algoritmo A* ya que este obtiene el camino óptimo. Dijkstra también obtiene el camino óptimo, pero el número de nodos que expande en general es mayor que en A*, por lo que para la competición es mejor utilizar A*. Por último, replanificamos la ruta cada vez que el mapa cambia, ya que el camino de la ruta antigua calculada podría contener (y suele contener) muros o trampas, por lo tanto el agente moriría o podría no llegar a la meta. Detectamos este tipo de casillas cuando el número de muros y trampas aumenta.

El pseudocódigo de la lógica seguida podría ser el siguiente:

```
si no hay camino construido o ha cambiado el mapa:
    actualizamos los muros y trampas del mapa
    construimos la nueva ruta llamando a la busqueda del AStar
ejecutamos la siguiente accion
```

3 Preguntas

Describa un caso para el que sería más recomendable/beneficioso utilizar el algoritmo de Dijkstra en lugar de A*, y viceversa.

Cuando intentamos encontrar la ruta más corta en un grafo cuyos pesos son negativos, el algoritmo A* podría caer en ciclos infinitos, mientras que Dijkstra no presenta este comportamiento. También es mejor utilizar Dijkstra siempre que la heurística utilizada en A* no estime bien el costo de un nodo hacia el nodo objetivo (por ejemplo, si subestima el costo).

En el resto de casos, es mejor utilizar el algoritmo A* ya que al tener un componente heurístico expande menos nodos.

En nuestro problema, por ejemplo, Dijsktra nunca será mejor que A*; puede conseguir dependiendo del mapa el mismo número de nodos expandidos, pero nunca superar A*.

¿Cuáles son las principales diferencias entre RTA* y LRTA*? En la práctica, ¿en qué afectan estas diferencias a la resolución del problema?

La principal diferencia entre RTA* y LRTA* es que, en la búsqueda, a la hora de actualizar el valor de la heurística del nodo actual, LRTA* actualiza al máximo entre el valor de la heurística actual y, entre los vecinos del nodo, el valor heurístico más pequeño, y RTA* actualiza al máximo entre el valor de la heurística actual y, entre los vecinos del nodo, el segundo valor heurístico más pequeños (si este existiese). Es decir:

- RTA*: $h(x) = \max(h(x), 2^{\text{o}}\min(c(x,y) + h(y)))$, donde $y \in vecinos(x)$.
- LRTA*: $h(x) = \max(h(x), \min(c(x,y) + h(y)))$, donde $y \in vecinos(x)$.

En la práctica, la principal diferencia es que RTA*, al actualizar al segundo mínimo, puede sobrestimar el valor heurístico de un nodo y por lo tanto escapar de mínimos locales más rápido, haciendo que el tiempo de ejecución sea menor que en LRTA*. Además, esto hace que RTA* no garantiza que encuentre el camino óptimo. Por el contrario, LRTA* aproxima mejor el valor heurístico del nodo a su valor heurístico real, garantizando encontrar el camino óptimo.

¿Cómo aplicaría el algoritmo de Dijkstra y el algoritmo A* para la resolución del juego "Labyrinth extended"?

Lo aplicaría del mismo modo que he implementado el agente de la competición. En la primera iteración, construimos el camino a la meta, y lo recorremos. En

el momento que pisemos una casilla oculta (y esto lo detectamos, por ejemplo, cuando el número de `immovablePositions` aumente), actualizamos las casillas no permitidas del mapa y realizamos otra llamada al método de búsqueda, actualizando además la posición inicial a la posición en que se encuentra el agente. De este modo, estaríamos ejecutando la búsqueda hasta el objetivo y replanificando dicha ruta cada vez que el mapa cambia, evitando así los posibles muros y trampas y garantizando que llegamos al objetivo.

Realice un script que, dado un mapa, genere uno nuevo posicionando al avatar en una localización distinta (siempre que en ésta no haya muros, trampas o casillas objetivo). Usando este script, ejecute el algoritmo RTA* múltiples veces sobre el mapa pequeño de labyrinth (original), de forma que la función $h(n)$ sea inicializada con los valores almacenados al final de la ejecución anterior. Finalmente, represente usando un mapa de calor (heatmap), el valor de $h(n)$ al finalizar la última ejecución de RTA*. Genere un segundo heatmap repitiendo el proceso con LRTA*. Por último, calcule el coste mínimo $h^*(n)$ para todas las casillas de este mapa usando A* (puede usar el script anterior) y represente esta función en un tercer heatmap. ¿En qué se diferencian los heatmaps obtenidos? ¿Qué conclusión se puede sacar de estos resultados?

Para realizar este ejercicio, he creado tres ficheros nuevos, correspondientes a los tres agentes a utilizar por el ejercicio. La clase es exactamente la misma estructura que el respectivo agente normal, solo que la estructura de datos utilizada para almacenar la heurística en cada caso la he declarado como `static` para que se utilicen los valores heurísticos anteriores, y para poder pintar el heatmap.

En el caso de A* y LRTA* el heatmap obtenido debería ser el mismo (es decir, el coste mínimo para llegar al objetivo de cada casilla del mapa), y el heatmap de RTA* es distinto. Esto se debe a que, en RTA*, al actualizar al segundo mínimo, este no garantiza encontrar el camino óptimo al objetivo, mientras que LRTA* y A* si garantizan encontrar el camino óptimo.

Nota: Los heatmaps no los he implementado correctamente, aún así lo he intentado y he intentado responder la pregunta según lo que debería salir.

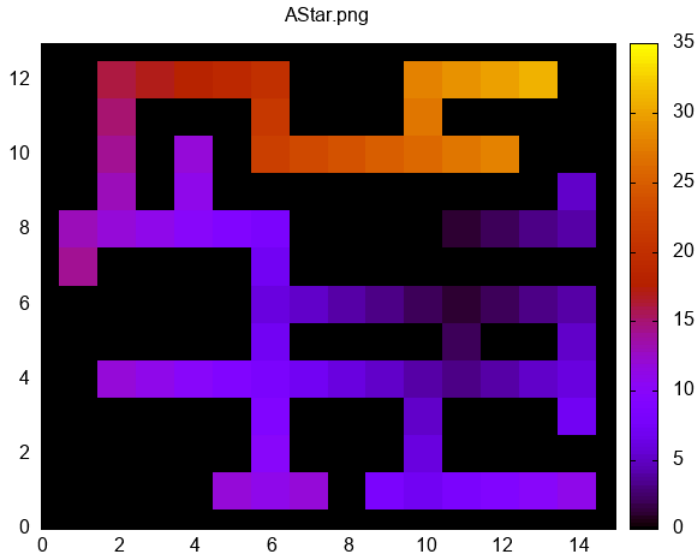

Figure 2: Heatmap A*

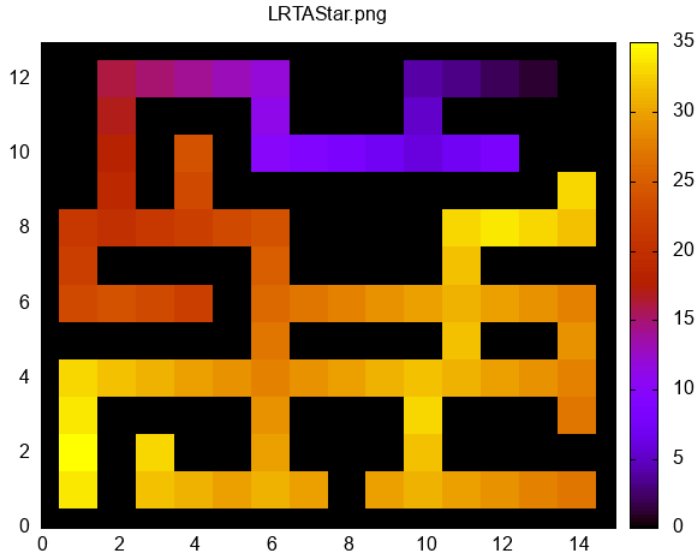

Figure 3: Heatmap LRTA*

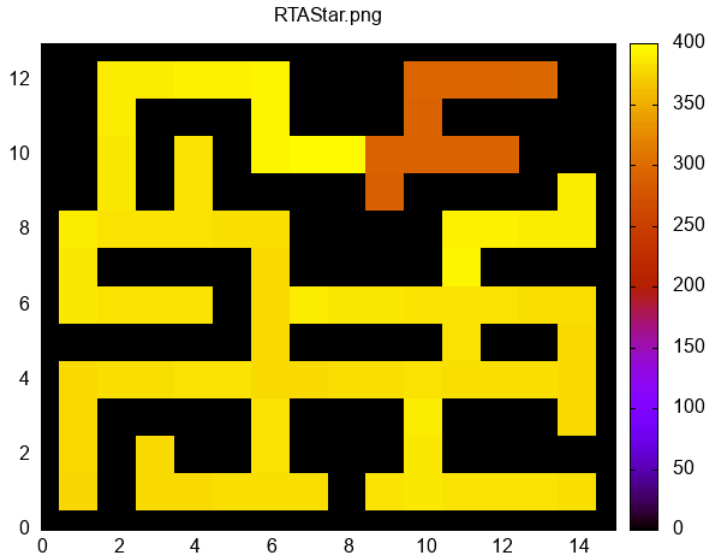

Figure 4: Heatmap RTA*