Técnicas de los Sistemas Ingeligentes (2021-2022)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 1:Experimentación con Técnicas de Búsqueda

Fernando Luque de la Torre

17 de abril de 2022

Índice

1. Tabla de resultados

Como introducción a los resultados obtenidos, debo aclarar varias cuestiones.
En primer lugar, aclarar que, para el algoritmo de búsqueda en anchura, el máximo número de nodos en memoria lo he calculado en cada momento como la suma entre el número de nodos que ya he visitado más el número de nodos que se encuentran para ser expandidos en la cola toExpand. Sin embargo, con el objetivo de mejorar la eficiencia del algoritmo, la lista de nodos visitados no es más que una matriz booleana, por lo que el número de nodos en memoria se reduciría a 3, que son la máxima cantidad de nodos que se van a almacenar en la cola toExpand, ya que son la máxima cantidad de vecinos que un nodo puede generar. Con esto quiero aclarar que el hecho de que haya 97 nodos en memoria por ejemplo, no significa que haya 97 objetos Node en memoria, sino que de la mayoría, únicamente almaceno un bit que nos dice si lo he visitado o no y del resto (que han sido generados como vecinos pero no visitados aún, sí que almaceno la información completa).
En DFS, el consumo máximo de memoria es el que se indica en las diapositivas de repaso, el número de nodos visitados.
En cuanto al algoritmo A*, con el objetivo de mejorar la eficiencia temporal, la lista de nodos abiertos se almacena duplicada. Por un lado, tenemos una cola con prioridad que nos extraerá los nodos en el orden indicado. Por otro, una matriz de Node, con el objetivo de agilizar las consultas a cerca de los nodos abiertos, ya que las consultas en las colas son muy poco eficientes. Además, la lista de cerrados también es una matriz, de nuevo para hacerlo más eficiente en tiempo. Sin embargo, la lista de cerrados no es necesario mantenerla ordenada por lo que esta no está duplicada. Aclaro esto ya que en el consumo máximo de memoria, no estoy contando el número de nodos en abiertos de forma duplicada, sino solo una vez, como si únicamente tuviera la cola con prioridad y sobre esta se realizaran las consultas (tal y como se explica en el pseudocódigo de las diapositivas de repaso).

Alg	Mapa	Runtime(ms)	Ruta (nodos)	Expandidos	En memoria
BFS	5	0,4206	16	96	97
	6	0,4246	35	130	131
	7	1,7625	111	846	846
	8	15	210	4630	4633
DFS	5	0,2865	28	71	71
	6	0,4993	65	81	81
	7	0,6712	239	402	402
	8	5,2549	936	1213	1213
A*	5	0,5299	16	39	57
	6	2,3158	35	85	104
	7	3,7345	111	716	735
	8	20,7402	210	3606	3702
IDA*	5	1,336	16	457	15
	6	1,0707	35	432	34
	7	1976,7324	111	20134917	110
	8	—	—	—	—
RTA*	5	4,1462	249	249	88
	6	2	138	138	83
	7	9,5857	702	702	401
	8	—	—	2001 (TICKS LIMIT)	1026

2. Cuestiones

2.1. Cuestión 1

Entre BFS y DFS, ¿qué algoritmo puede ser considerado más eficiente de cara a encontrar el camino óptimo?
En general, la búsqueda en anchura (BFS) encontrará un camino más óptimo que la búsqueda en profundidad (DFS) ya que esta segunda puede dar muchos pasos innecesarios dependiendo simplemente del orden en que se expanden los nodos, como pasa por ejemplo en el laberinto número 8.
Sin embargo, hay que tener en cuenta que el tratar de conseguir una solución mejor implica un sacrificio en eficiencia en memoria, consumiendo el BFS más que el DFS.

2.2. Cuestión 2

¿Se podría decir que A* es más eficiente que BFS?
En cuanto a la calidad de la solución encontrada, en general, para un problema de búsqueda de caminos, A* siempre da el camino más óptimo, mientras que BFS no asegura el camino más óptimo en grafos ponderados. Sin embargo, en nuestros casos, como el coste de desplazarse de una casilla a otra es constante (el grafo no es ponderado), el algoritmo BFS es capaz de encontrar el camino óptimo.

En cuanto a eficiencia temporal podemos observar que aunque A* expanda siempre un menor número de nodos, la mayor complejidad de los cálculos que hay que realizar para expandir cada nodo, hace que la eficiencia de A* no sea siempre menor que la de BFS.
Por último, como ya he comentado, A* expande siempre un menor número de nodos que BFS por lo que el consumo de memoria es siempre menor.

2.3. Cuestión 3

¿Cuáles son las principales diferencias entre A* e IDA*? ¿En qué contextos es más conveniente usar uno u otro?
La principal diferencia entre IDA* y A* es la introducción de una cota al algoritmo de A* que limita la utilización de memoria. Es por eso que, en tiempo, el algoritmo de IDA* es mucho menos eficiente que el de A* pero sería muy adecuada su utilización en situaciones concretas en las que la utilización de poca memoria fuera un aspecto crucial y la eficiencia temporal no lo fuera tanto.

2.4. Cuestión 4

¿Se podría decir que RTA* es más eficiente que A*?
RTA* es un algoritmo de búsqueda en tiempo real, que es capaz de funcionar en situaciones de incertidumbre. Por otro lado, A* es un algoritmo de búsqueda offline, que necesita de información global para funcionar.
Es por eso que no sería justo realizar una comparación directa entre A* y RTA* pues la información heurística global que utiliza A* hace que este sea mucho más eficiente con respecto a su correspondiente algoritmo de tiempo real RTA*.
Por tanto, no son algoritmos que compitan entre sí, sino que habrá determinadas situaciones en los que sea necesario utilizar RTA* porque las circunstancias lo necesiten y otras en las que podamos obtener información global que nos permita utilizar A*.