Técnicas de los Sistemas Inteligentes

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

Curso 2021-2022

Pedro Bedmar López - 75935296Z
pedrobedmar@correo.ugr.es
Grado en Ingeniería Informática

---

1. Tabla de resultados

El objetivo de esta práctica es comparar el funcionamiento de 5 algoritmos de búsqueda: BFS, DFS (ambos de búsqueda no informada), A*, IDA* y RTA* (de búsqueda heurística o informada). Para ello, hemos tomado los pseudocódigos proporcionados por los profesores y los hemos implementado en Java.

Construimos esta tabla con los datos de ejecución de los algoritmos sobre cuatro mapas de distinto tamaño, donde se mide el tiempo de ejecución acumulado, el tamaño de la ruta calculada (o número de acciones que realiza el agente), el número de nodos expandidos (nodos para los que se ha comprobado si son nodos objetivo) y el máximo número de nodos en memoria.

Alg.	Mapa	Runtime (ms)	Tamaño de la ruta calculada	Num. de nodos expandidos	Max. num. de nodos en memoria
BFS	Muy pequeño	0.41613	15	96	97
	Pequeño	0.58643	34	130	131
	Mediano	2.19621	110	846	846
	Grande	4.63245	209	4632	4635
DFS	Muy pequeño	0.21244	27	71	71
	Pequeño	0.23764	64	81	81
	Mediano	0.81093	238	402	402
	Grande	1.72530	935	1213	1213
A*	Muy pequeño	0.49926	15	40	57
	Pequeño	1.25565	34	86	104
	Mediano	3.73479	110	717	735
	Grande	12.85051	209	3607	3702
IDA*	Muy pequeño	2.02634	15	133	16
	Pequeño	3.35277	34	427	35
	Mediano	TO	110	484650	111
	Grande				
RTA*	Muy pequeño	1.28131	53	53	46
	Pequeño	1.35061	38	38	55
	Mediano	7.30048	1594	1594	561
	Grande	8.47751	1567	1567	631

2. Cuestiones a responder

2.1. Entre BFS y DFS, ¿qué algoritmo puede ser considerado más eficiente de cara a encontrar el camino óptimo?

BFS es un algoritmo que recorre el árbol donde se aplica en anchura. Por tanto, si los costes asociados a cada arista del árbol son iguales, encontrará el camino óptimo al nodo objetivo. Esto ocurre porque si todos los pesos de las aristas son iguales, lo importante para que un camino sea óptimo es la longitud de éste: y al recorrer el árbol en anchura, el primer camino al nodo objetivo que vamos a descubrir coincide con el camino más corto.

En cambio, en DFS no ocurre esto ni aunque el coste de desplazamiento entre nodos sea constante, ya que recorremos el árbol en profundidad. Podríamos encontrar primero una rama del árbol que llevara al nodo objetivo pero que no fuese la de menor longitud, y que por tanto se detuviese el algoritmo devolviendo una solución que no es la óptima.

Por tanto, aunque el tiempo de ejecución, el número de nodos expandidos y el consumo de memoria sean menores para DFS que para BFS, el primero no nos garantiza el camino óptimo, cosa que BFS sí (ya que para este problema el coste de desplazamiento entre nodos es constante) y es el algoritmo a elegir si necesitamos encontrar el camino óptimo.

2.2. ¿Se podría decir que A* es más eficiente que DFS?

Depende. Si necesitamos encontrar el camino óptimo a un nodo solución, ya hemos visto que DFS no nos asegura encontrarlo. En cambio A* sí lo encuentra, con una condición: que la heurística que se utilice para guiar al algoritmo sea admisible. En nuestro caso lo es, ya que $h(n) \leq h^*(n)$ para todas las n casillas del tablero.

En cuanto al número de nodos expandidos, DFS expande un mayor número de nodos que A*. Esto se produce porque el primero expande los nodos en un orden que no es inteligente, sin priorizar aquellos que tienen más probabilidad de pertenecer al camino óptimo hasta la solución. Por otro lado, A* es guiado por una heurística, priorizando aquellos con un valor de $f = g + h$ más bajo. Por tanto, la calidad de la solución obtenida por A* depende de la calidad de la heurística. Es también por esta heurística por la que A* consume menos memoria, ya que evita expandir nodos que difícilmente vayan a pertenecer al camino óptimo.

En cuanto al tiempo de ejecución ocurre al revés, ya que aunque A* necesite expandir menos nodos, utiliza una cola con prioridad (implementada como un heap) para ordenarlos según el valor de f. DFS no utiliza esta estructura, usa recursividad. Es más costoso en términos de tiempo realizar operaciones sobre la cola con prioridad, especialmente eliminar un nodo n ($O(n)$) o insertarlo ($O(log(n))$), por lo que A* tiene un runtime mayor.

2.3. ¿Cuáles son las principales diferencias entre A* e IDA*? ¿En qué contextos es más conveniente usar uno u otro?

Aunque ambos algoritmos encuentran el camino óptimo, A* fue propuesto en primer lugar. IDA* se desarrolló con el propósito de reducir la memoria utilizada por A*, utilizando una búsqueda acotada sobre el grafo. Es por esto que el máximo número de nodos en memoria es considerablemente menor que en A* en todos los mapas en la tabla. En cada momento sólo almacena el camino desde el nodo inicial hasta el actual, en vez de mantener una cola de abiertos y cerrados como A*.

Por otro lado, el número de nodos expandidos y el tiempo de ejecución aumentan. IDA* utiliza una cota para limitar la búsqueda a una determinada profundidad del grafo. Si en esa profundidad no se ha encontrado la solución, se aumenta la cota y se comienza la búsqueda desde cero, volviendo a recorrer los nodos y caminos visitados en la iteración anterior. El no mantener un control sobre qué nodos se han visitado anteriormente, como realiza A*, hace que el tiempo de ejecución aumente considerablemente, repitiendo búsquedas en caminos que ya se habían comprobado.

Si en nuestro propósito encontramos limitaciones de memoria (por ejemplo, estamos utilizando un sistema embebido), será más conveniente utilizar IDA*. En cambio, si no tenemos restricciones de memoria y queremos tener unos tiempos de ejecución menores, A* es nuestra opción. Como se aprecia en la tabla, a partir de mapas no demasiado grandes IDA* es impracticable en tiempo, dando lugar a un timeout. Aún así, ambos algoritmos aseguran encontrar el camino óptimo en nuestro problema.

2.4. ¿Se podría decir que RTA* es más eficiente que A*?

Cuando nos encontramos ante una tarea de búsqueda donde tenemos restricciones de tiempo muy fuertes, es necesario reconsiderar el uso de A*. Este algoritmo, como todos los que hemos visto hasta ahora, calcula el camino hasta el nodo solución antes de realizar ningún movimiento real en el mapa. En cambio, RTA* sólo hace cálculos para el siguiente movimiento que va a realizar, distribuyendo el tiempo de ejecución entre todos ticks y evitando timeouts en el primero. O sea, RTA* sólo expande un nodo y realiza un movimiento por tick.

Sobre su optimalidad, no asegura encontrar el camino óptimo. Esto se debe a que sólo es capaz de razonar en función del vecindario próximo al nodo actual. En la búsqueda se ayuda de una heurística, y va actualizando su valor conforme recorre el mapa. De hecho, puede visitar el mismo nodo en varias ocasiones, actualizando su valor heurístico. Estas múltiples visitas al mismo nodo harán que el tamaño de la ruta y el número de nodos expandidos en RTA* sean mayores que en A*.

En memoria sólo se almacenan los nodos que se han visitado, por lo que necesita almacenar menos información que A*. En definitiva, RTA* es más eficiente en memoria y tiempo (distribuyendo los cálculos entre ticks) que A*, pero cuenta con la desventaja de no obtener el camino óptimo, con caminos que según el mapa pueden ser mucho más largos.