Inteligencia Artificial
Practica 1: Búsqueda no informada

Memoria

Dorian Boleslaw Wozniak (817570@unizar.es)

Resultados obtenidos

Problema	Profundidad	Expand	Q.Size	MaxQS	tiempo
BFS-G-3	3	5	4	5	30
BFS-T-3	3	6	9	10	3
DFS-G-3	59123	120491	39830	42913	2043
DFS-T-3	---	---	---	---	(2)
DLS-9-3	9	10	0	0	1
DLS-3-3	3	4	0	0	1
IDS-3	3	9	0	0	1
UCS-G-3	3	16	9	10	3
UCS-T-3	3	32	57	58	1
BFS-G-9	9	288	198	199	7
BFS-T-9	9	5821	11055	11056	35
DFS-G-9	44665	141452	32012	42967	1225
DFS-T-9	---	---	---	---	(2)
DLS-9-9	9	5474	0	0	13
DLS-3-9	0	12	0	0	0
IDS-9	9	9063	0	0	89
UCS-G-9	9	385	235	239	7
UCS-T-9	9	18070	31593	31594	292
BFS-G-30	30	181058	365	24048	1430
BFS-T-30	---	---	---	---	(2)
DFS-G-30	62856	80569	41533	41534	1848
DFS-T-30	---	---	---	---	(2)
DLS-9-30	0	4681	0	0	7
DLS-3-30	0	9	0	0	0
IDS-30	---	---	---	---	(1)
UCS-G-30	30	181390	49	24209	1072
UCS-T-30	---	---	---	---	(2)

Búsqueda en anchura (BFS)

Lo más importante a destacar de la búsqueda en anchura es el crecimiento exponencial de los estados expandidos conforme la profundidad del camino al objetivo mas corto del problema aumenta. La complejidad temporal y espacial aumenta exponencialmente respecto a la profundidad de la solución mínima, al expandir los nodos nivel a nivel. En el peor de los casos, tiene que expandir todos los nodos hasta el nivel donde se encuentra el objetivo, y debe almacenar una gran cantidad de nodos frontera de cada subárbol explorado. Esto acaba causando que la búsqueda en anchura en árbol, que a diferencia de la búsqueda en grafo puede repetir estados, incluso si expandir cada nodo fuera eficiente la memoria requerida para almacenar el árbol entero es excesiva (todos los movimientos son reversibles, por lo que hay muchas estructuras repetidas). Aun así, en el caso de que acabe se garantiza que la solución es completa (se puede resolver siempre en un número finito de pasos y hay máximo 4 movimentos válidos por estado, 3 en los bordes y 2 en las esquinas) y óptima (el primer nodo objetivo encontrado siempre es el más somero en el árbol).

Búsqueda en profundidad (DFS)

En la búsqueda primero en profundidad, las búsquedas en árbol acaban encontrando una solución, pero en un número de movimientos absurdamente grande (40000 - 60000). Esto se debe a la forma en la que explora el árbol de búsqueda: en vez de explorar para cada estado todos los movimientos posibles, el algoritmo en profundidad elige el nodo más profundo y lo expande, eligiendo a su vez el primer nodo de la frontera del nodo recien expandido al ser ahora el de mas profundidad. Esto convierte al algoritmo en uno de, esencialmente, fuerza bruta donde va probando estados del puzle no visitados hasta encontrar alguna solución. Esto da a situaciones como que la búsqueda encuentra antes la solución del tablero de 9 movimentos que el de 3 movimientos. Esto no funciona en absoluto con la búsqueda en árbol: al no tener en cuenta nodos ya expandidos, es fácil que entre en búcles, y continuamente introduce nodos nuevos en la frontera hasta quedarse sin memoria.

Búsqueda en profundidad recursiva limitada (DLS)

La búsqueda en profundidad limitada es una búsqueda de profundidad pero solo hasta una capa determinada: los nodos en la profundidad límite son tratados como nodos sin sucesores. Esto evita los bucles infinitos, pero no garantiza una solución. El método de búsqueda normalmente se utiliza cuando ya se conoce la profundidad de una posible solución. Para el tablero de 3 movimientos, encuentra la solución óptima. En el de 9 pasos la búsqueda limitada a profundidad 9 acaba, efectivamente, encontrando la solución que está a profundidad 9. El resto de casos, el tablero de 9 con límite de 3 y el de 30 en ambos casos, el límite es demasiado pequeño para encontralos, asi que siempre llega al *cutoff* sin encontrar solución. Además, al ser la implementación recursiva, no hay ni cola ni pila para almacenar una frontera: los nodos se van expandiendo con llamadas recursivas a medida que el algoritmo se los encuentra.

Búsqueda en profundidad iterativa limitada (IDS)

La búsqueda en profundidad iterativa realiza búsquedas de profundidad aumentando el límite cada vez. Es un compromiso entre los algoritmos de busqueda en anchura y en profundidad. Acaba encontrando la solución óptima pero sin ocupar tanto espacio en memoria como el BFS, y además puede encontrar la profundidad ideal para una búsqueda en profundidad limitada no iterativa. Sin embargo, esto significa que, según la profundidad de la solución, deberá hacer una búsqueda en profundidad para cada nuevo límite. Esto causa problemas pues el tiempo requerido para cada búsqueda en profundidad en profundidad aumenta exponencialmente segun el límite. En el problema de 30 movimientos esto causa que no acabe en tiempo razonable ya que debe hacer la búsqueda para el límite en 1, 2, 3, 4, ..., 27, 28, 29 hasta llegar a 30, tomando cada uno mucho mas tiempo que el anterior.

Búsqueda de coste uniforme (UCS)

Teniendo en cuenta que los movimientos en principio tienen el mismo coste todos, el algoritmo de coste uniforme debería comportarse en principio de la misma manera que el algoritmo de búsqueda en anchura. Aunque los problemas y comportamiento es parecido, parece tardar mas en completarse, y explora más estados que la búsqueda en anchura. Esto podría deberse a la implementación concreta de este algoritmo y como determina el órden en el que encola los siguientes nodos de la frontera. También falla en la búsqueda en grafo para el árbol de 30 movimientos por memoria, al tener que almacenar demasiados estados en la frontera, muchos repetidos.