INTELIGENCIA ARTIFICIAL

Práctica 2

Curso 2018-2019

Autor:
Rubén Rodríguez Esteban 737215

1. Resumen

Esta práctica tiene como objetivo familiarizarse con el código en java para resolver problemas de búsqueda informada y diferenciar sus características.

2. Introducción

En esta práctica se ha trabajado modificando el código java con el que se trabajó en la práctica anterior. El trabajo ha consistido en realizar experimentos y recopilar información relevante de la búsqueda relacionada con aspectos de eficiencia. Realizaremos con el 8-puzzle primero búsquedas ciegas (BFS e IDS) y búsquedas informadas A* con las heurísticas de fichas descolocadas y Manhattan. En esta práctica vamos a utilizar la búsqueda A*, y a comparar la eficiencia de los algoritmos mostrando el número de nodos generados y el factor de ramificación efectivo b* para distintas profundidades.

.3. Métricas incorporadas

Se ha incluido en las métricas el número de nodos generados. Dicha métrica se ha incorporado en la clase *IterativeDepeningSearch.java* y en la clase *NodeExpander.java*. Todas las modificaciones realizadas aparecen comentadas de manera clara en los ficheros anteriores.

Paralelamente se ha añadido una nueva clase en la librería *aima.core.util.math* denominada *bisección.java.* Esta clase cuenta con un conjunto de métodos que permiten obtener los ceros de la función N = b* (b*d - 1) / (b* - 1) por aproximaciones sucesivas, donde *b** es el factor de ramificación efectivo, *N* el número de nodos generados y *d* la profundidad de la solución.

.3. Análisis de experimentos

Se han realizado cien experimentos aleatorios de la profundidad deseada y, posteriormente, se ha procedido a calcular la media de los nodos anteriormente generados. Para ello se ha empleado una clase llamada *GenerateInitialEightPuzzleBoard.java*, suministrada para que generar aleatoriamente estados iniciales, y estados finales de la profundidad deseada. Para poder generar estados iniciales aleatorios se ha empleado el método *randomIni()*, definido en fichero anterior, y para crear los estados finales, se ha empleado el método random() que permite generar estados objetivo aleatorios a partir de un estado inicial aleatorio y una profundidad concreta, ambos pasados como parámetros.

En la realización de los experimentos se ha tenido en cuenta que la solución obtenida puede no ser la deseada dado que se hayan dado *d* pasos al estado final, no garantiza que no haya caminos más cortos al estado final. Por lo que al efectuar el experimento se ha comprobado que la solución óptima es de la profundidad deseada.

Para realizar esta tarea se han procedido a realizar ciertas modificaciones en las clases. Concretamente se han creado la clase *EightGoalTest2.java* para almacenar el estado objetivo definido de forma aleatoria, y dos heurísticas para llevar a cabo las búsqedas informadas. Dichas heurísticas son la heurística de Manhattan y la heurística de piezas descolocadas, definidas respectivamente en los ficheros *ManhattanHeuristicFunction.java*, y MisplacedTilleHeuristicFunction para que sean útiles para cualquier estado final. Todas las operaciones y cambios realizados están documentados en el código java anterior.

Una vez realizados todos los pasos anteriores se ha creado un fichero donde se han realizado los cien experimentos con una profundidad variante entre una cota inferior y una superior. Dicho fichero se denomina *EightPuzzleInfoSearch.java*. A continuación se muestra la salida obtenida en terminal al ejecutar el programa anterior.

	NODOS GENERADOS				b*			
d	BFS	IDS	A*h(1)	A*h(2)	BFS	IDS	A*h(1)	A*h(2)
2	7	10	5	5	2,19	2,70	1,79	1,79
3	18	31	9	8	2,22	2,75	1,66	1,58
4	37	99	12	11	2,13	2,84	1,49	1,45
5	70	260	17	14	2,06	2,79	1,44	1,37
6	127	745	25	18	2,00	2,80	1,43	1,33
7	217	2200	35	23	1,95	2,82	1,41	1,30
8	357	6235	49	28	1,90	2,82	1,40	1,28
9	616	16566	76	34	1,88	2,80	1,42	1,26
10	1031	50235	116	48	1,85	2,83	1,43	1,28
11	1661	----	175	62	1,83	----	1,44	1,27
12	2737	----	259	86	1,81	----	1,44	1,28
13	4450	----	405	127	1,79	----	1,45	1,30
14	6963	----	636	158	1,77	----	1,46	1,29
15	11292	----	985	231	1,76	----	1,47	1,31
16	17396	----	1520	302	1,75	----	1,47	1,31
17	27084	----	2394	388	1,73	----	1,48	1,30
18	41058	----	3560	550	1,72	----	1,48	1,31
19	63301	----	5451	711	1,71	----	1,48	1,31
20	91173	----	9218	986	1,69	----	1,49	1,31
21	129193	----	12686	1316	1,68	----	1,49	1,32
22	174241	----	21068	1639	1,66	----	1,50	1,31
23	227162	----	30403	2344	1,64	----	1,49	1,32
24	286039	----	46769	3116	1,62	----	1,49	1,32

De acuerdo con la salida mostrada por el programa podemos observar que los algoritmos de búsqueda ciega o no informada son menos eficientes que los algoritmos de búsqueda informada, a pesar de que todos los algoritmos ejecutados son de complejidad exponencial tanto en tiempo como en espacio. Ésto se puede observar por ejemplo en la métrica que calcula el número de nodos generados, ya que existe una diferencia sumamente notable entre los resultados obtenidos por ambos tipos de algoritmos, sobre todo, conforme aumenta el grado de profundidad en el espacio de búsqueda. Sin embargo, se debe tener en cuenta que cuanto mejor y más consistente sea la heurística seleccionada en los algoritmos de búsqueda informada, menor será la complejidad del algoritmo.

La razón por la que se produce este aumento es debida a que en los algoritmos de búsqueda ciega no existe ningún tipo de función de evaluación o heurística, por lo que no existe ningún criterio que permita redirigir la búsqueda para minimizar el número de nodos visitados, por lo que siempre va a ser mayor. El algoritmo de A* siempre va a obtener mejores resultados que los algoritmos BFS e IDS dado que en los algoritmos A* se tiene en cuenta para expandir los nodos tanto el valor de la heurística como el coste de la acción, es decir, se tienen en consideración ambos factores para seleccionar el nodo a expandir. Sin embargo, el algoritmo BFS a la hora de expandir los nodos no tiene en cuenta los costes de las acciones sino el orden en el que han sido insertados en la frontera, y de igual modo ocurre con el algoritmo IDS, de ahí, que el algoritmo A* obtiene siempre mejores resultados.

Adicionalmente, también se puede observar que en todos los experimentos se ha encontrado una solución. La justificación de este hecho estriba es que los algoritmos A*, BFS e IDS son completos, es decir, si existe una solución la van a encontrar, por lo que la profundidad podría incrementarse sin peligro de que los algoritmos se queden bloqueados por el camino. Sin embargo, el algoritmo IDS con niveles grandes tarda mucho más que los restantes, es decir, encuentra la solución pero en un tiempo sumamente grande.

También se puede observar que la bisección aumenta conforme se incrementa el grado de profundidad. Ésto es debido a que al aumentar la profundidad de búsqueda paulatinamente también se incrementa el número de nodos visitados, por lo que también aumenta la bisección.