Práctica 2
Inteligencia Artificial

Universidad de Zaragoza

Marcos Ruiz García, 648045

1 de noviembre de 2015

Índice

1. Introducción

Se nos ha encargado realizar experimentos y recopilar información relevante de la búsqueda relacionada con aspectos de eficiencia. Dichos experimentos se realizaran con el problema del 8-puzzle: En primer lugar búsquedas ciegas (BFS e IDS) y posteriormente búsquedas informadas A* con las heurísticas de fichas descolocadas y Manhatan. De esta manera, se podrán comparar la eficiencia de los algoritmos mostrando el número de nodos generados y el factor de ramificación efectivo b* para distintas profundidades.

2. Tareas

2.1. Implementación de la clase Biseccion

La clase Biseccion.java se ha copiado en aima.core.math ya que, en este caso, se nos da hecha. Dicha clase permite obtener los ceros de la función (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.

$$N = b^*(b^{*d} - 1)/(b^* - 1) \tag{1}$$

Todas las veces se invoca el método

```
metodoDeBiseccion(double a, double b, double error)
```

con a = 1.0001, b = 4.0 y error = e^{-10}. Se ha elegido este intervalo [a,b] debido a que el mínimo factor de ramificación que se puede conseguir es 1, ya que uno sería ir directo a la solución, y el máximo al que se ha llegado es 3 en todos los experimentos realizados.

2.2. Generación de 100 experimentos aleatorios de la profundidad deseada

Para la realización de los 100 experimentos se ha creado la clase EightPuzzleDemoPrac2.java la cual realiza 100 pruebas por cada nivel de profundidad y cada algoritmo de búsqueda, y guarda la suma de todos los resultados en una serie de tablas. Finalmente, para mostrar la media de los resultados, se divide lo guardado en dichas tablas por 100 y se muestran por pantalla. Para asegurarnos que las pruebas realizadas eran de la profundidad adecuada y no se encontraba la solución antes de lo que suponíamos, se comprueba la profundidad de la solución encontrada con la que se esperaba y en el caso de no coincidir, dicha prueba se ignoraría en todos los algoritmos de búsqueda.

2.3. Reescribir las heurísticas

Se nos ha encargado adaptar MisplacedTilleHeuristicFunction.java y ManhattanHeuristicFunction.java de manera que acepten cualquier estado final, es decir, hay que hacerlos más genéricos para poder realizar las pruebas lo más aleatoriamente posible. Ambas heurísticas son admisibles ya que nunca sobreestiman el coste real.

2.3.1. MisplacedTilleHeuristicFunction

Esta heurística cuenta el número de fichas mal colocadas.

2.3.2. ManhattanHeuristicFunction

Esta heurística suma el número de posiciones a la que está cada ficha de su posición objetivo. Como se puede observar en la sección de los resultados, esta heurística es mucho mejor que la anterior ya que nos da un número más cerca de la realidad, es decir, más alto.

3. Resultados

		Nodos Generados					b*			
	d	BFS	IDS	A*h(1)	A*h(2)		BFS	IDS	A*h(1)	A*h(2)
	2	8	10	5	5		2,36	2,79	1,88	1,88
	3	18	32	9	8		2,20	2,77	1,70	1,61
	4	38	100	12	11		2,15	2,83	1,51	1,46
	5	68	269	17	14		2,04	2,78	1,44	1,37
	6	121	761	24	18		1,98	2,78	1,41	1,33
	7	218	2206	32	22		1,94	2,80	1,38	1,29
	8	382	6565	51	28		1,92	2,82	1,40	1,28
	9	629	17647	71	34		1,88	2,81	1,40	1,25
	10	1019	49404	115	48		1,85	2,80	1,42	1,27
	11	1614	0	170	63		1,82	0,00	1,43	1,27
	12	2692	0	269	84		1,80	0,00	1,44	1,27
	13	4342	0	400	110		1,79	0,00	1,45	1,27
	14	6984	0	614	143		1,77	0,00	1,46	1,27
	15	11512	0	1002	216		1,76	0,00	1,47	1,29
	16	17626	0	1503	290		1,75	0,00	1,47	1,29
	17	27077	0	2410	393		1,73	0,00	1,48	1,29
	18	41025	0	3643	499		1,72	0,00	1,48	1,29
	19	62614	0	5619	719		1,71	0,00	1,48	1,30
	20	90112	0	9103	912		1,69	0,00	1,49	1,30
	21	128559	0	12757	1206		1,68	0,00	1,49	1,30
	22	172434	0	20871	1630		1,66	0,00	1,49	1,30
	23	231884	0	30559	2215		1,64	0,00	1,49	1,30

4. Conclusión

Como se puede observar en la sección Resultados tanto BFS como IDS, al ser búsquedas ciegas generan una cantidad de nodos mucho más altas que las búsquedas no ciegas, debido a esto podemos ver, que al añadir una heurística, por mala que sea, siempre será mejor que no aplicar nada. También se puede observar que la heurística ManhattanHeuristicFunction es mucho más eficiente que la heurística MisplacedTilleHeuristicFunction como era de esperar. En las dos heurísticas, llega un momento en el cual el factor de ramificación (b^*) se estabiliza y para de bajar lo cual nos hace ver de una manera más clara cual es el mejor algoritmo de búsqueda.