

INTELIGENCIA ARTIFICIAL

PRÁCTICA 2



Enrique Ballesteros Horcajo
Ignacio Iker Prado Rujas

5 de Noviembre de 2014

Índice

1.	Introducción: AimaDemoApp	1
2.	Puzle de 8 extremo	1
3.	AimaDemoApp: Desde Arad hasta Bucharest	1
4.	SearchDemoOsmAgentApp: Desde Altheim hasta Leibi	2
5.	Librerías y definiciones de estado para el puzzle de 8	2
6.	Librerías y definiciones de estado para el mapa	2
7.	Conclusiones	2

Petaqueo	del	bueno	every	day
Vale	4,8 ms	2 ms	0,7 ms	6 ms
Kike?	3,5 ms	2,7 ms	3 ms	5,8 ms

Cuadro 1: Esto es un gui3n de tabla.

Arad-Bucarest	Depth First	Breadth First	A*
Step 1	Timisoara - 118	Sibiu - 140	Sibiu - 140
Step 2	Lugoj - 229	Fagaras - 239	RimnicuVilcea - 220
Step 3	Mehadia - 299	Bucarest - 450	Pitesti - 317
Step 4	Dobreta - 374	-	Bucarest - 418
Step 5	Craiova - 494	-	-
Step 6	Pitesti - 632	-	-
Step 7	Bucharest - 733	-	-
Total	733 in 7 steps	450 in 3 steps	418 in 4 steps

Cuadro 2: Para la b3squeda de caminos en el mapa desde Arad hasta Bucarest, comparaci3n entre la b3squeda en profundidad, en anchura y el A* con SLD.

1. Introducci3n: AimaDemoApp

Para esta primera parte, tras haber importado el proyecto a Eclipse, hemos ejecutado el fichero `AimaDemoApp.java`, probando las distintas posibilidades que ofrece. Tiene dos modos: *Applications* y *Demos*, y la principal diferencia est3 en que en la primera disponemos de una interfaz gr3fica, 3til para comprender mejor qu3 esta haciendo el algoritmo correspondiente.

La primera aplicaci3n que podemos usar es para el coloreado de mapas mediante *backtracking* con distintas heur3sticas. Es muy interesante ejecutar la aplicaci3n paso a paso, para ver como se desarrolla el 3rbol. Adem3s, hay algunos juegos como el puzzle de 8, el 3 en raya, las *n*-reinas y el conecta 4. Los algoritmos que utilizan son los habituales, con algunas variaciones en la heur3stica: minimax, poda $\alpha - \beta$, b3squeda en anchura, b3squeda en profundidad, A*, voraz... Adem3s, incluye una aplicaci3n para encontrar el camino m3nimo entre dos nodos de un grafo, que aqu3 es un mapa de Ruman3a y otro de Australia.

Luego, en cuanto a las demos, aparecen algunas aplicaciones nuevas, pero creemos que tiene menos inter3s porque s3lo puedes ver el resultado final y tratar de interpretarlo, y no puedes modificar par3metros como en las aplicaciones anteriores, o incluso interferir en los juegos haciendo el movimiento que quieras.

2. Puzzle de 8 extremo

3. AimaDemoApp: Desde Arad hasta Bucharest

SLD es la heur3stica de l3nea recta, que elige la ciudad que est3 m3s cerca en l3nea recta del objetivo. En el cuadro adjunto podemos ver como se desarrolla el algoritmo en cada caso. Cabe destacar tambi3n que en el primero se expanden 10 nodos, mientras que en el segundo y en el tercero son 5.

El algoritmo de la b3squeda en anchura no encuentra el camino m3nimo porque

4. SearchDemoOsmAgentApp: Desde Altheim hasta Leibi

5. Librerías y definiciones de estado para el puzzle de 8

Para el problema del puzzle de 8 se utiliza un array de enteros de tamaño 9, sencillo pero efectivo. El rango de valores para el vector es $0 \leq i \leq 8$ (sin repeticiones), y de este modo el cero corresponde al hueco en el puzzle. El resto de casillas se corresponden 1-1 con su representación gráfica, leído de izquierda a derecha y de arriba a abajo. Además, se define en `EightPuzzleBoard.java` en el paquete `aima.core.environment.eightpuzzle` como array privado de enteros. También es importante notar que en función de desde donde se llame al constructor de esta clase, se pasa como parámetro un vector diferente (en función de la dificultad).



1	2	5
3	4	
6	7	8

Figura 1: Puzzle de 8 correspondiente a `state = new int[]{1, 2, 5, 3, 4, 0, 6, 7, 8}`.

En cuanto a los operadores que trabajan con el estado, *se centran en el hueco* y hay cuatro: `moveGapLeft()`, `moveGapUp()`, `moveGapRight()`, `moveGapDown()`. Hacen lo propio, tras comprobar que pueden hacer el movimiento pedido con `CanMoveGap()` calculan en qué posición está el hueco y cambian sus coordenadas en el mapa hacia la izquierda, arriba, derecha o abajo respectivamente. Para ello simplemente hacen un “*swap*”, guardando (por ejemplo para el primer caso) el valor situado a la izquierda del cero, poniéndolo a la derecha y escribiendo un cero a la izquierda. Es interesante ver como transforman un tablero 3×3 en un vector lineal de 9 posiciones. Dadas las coordenadas (x, y) la posición correspondiente en el vector es $3x + y$ (contando como $(1, 1)$ la esquina inferior izquierda donde está el 6 en la imagen), y así por ejemplo, en la figura vemos que 7 está en las coordenadas $(2, 1) \Rightarrow 3 \cdot 2 + 1 = 7$, luego `state[7] = 7`. Esto se consigue mediante las funciones `getAbsPosition()` y `setValue()`, en la misma clase que la comentada anteriormente.

En el paquete `aima.core.environment.eightpuzzle` se encuentra la clase donde se calcula el número de fichas descolocadas: `MisplacedTilleHeuristicFunction.java`.

6. Librerías y definiciones de estado para el mapa

7. Conclusiones

Bibliografía

- [1] Russell, S.; Norvig, P,
Artificial Intelligence, a modern aproach.
New Jersey: Pearson, 2010.
- [2] Apuntes y transparencias de Inteligencia Artificial,
Doble Grado Matemáticas - Ing. Informática, U.C.M., 2014-2015.