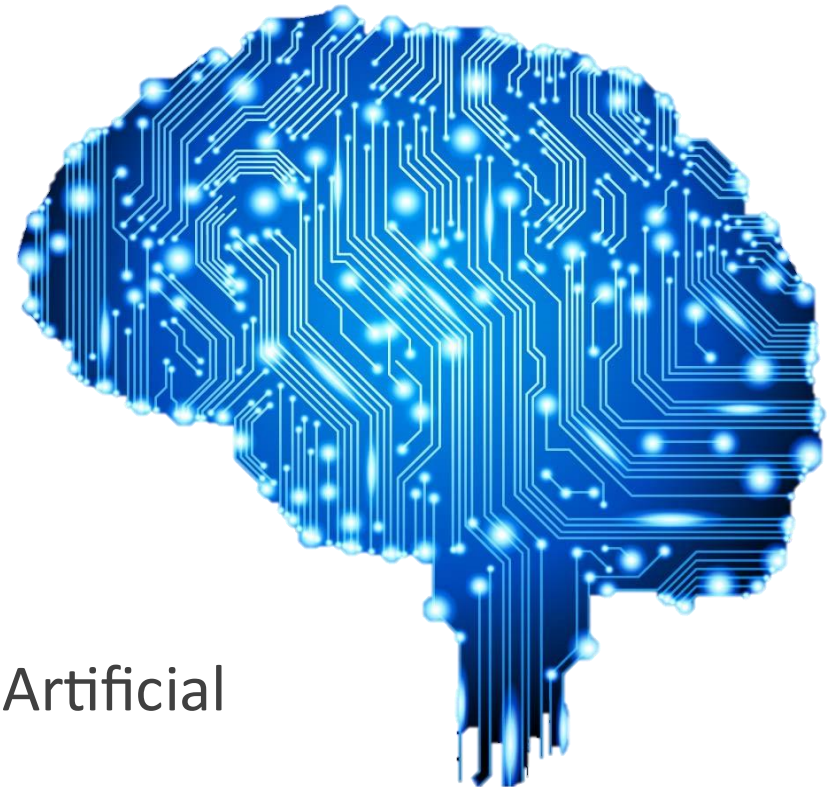
Representación y búsqueda en la librería AIMA



Inteligencia Artificial
2016/2017

Raúl Gil Fernández
Alberto Rodríguez – Rabadán Manzanares

1. AIMA Demo App: Eight Puzzle

	Coste del camino	Nodos expandidos	Tamaño de cola	Máximo tamaño de cola	Tiempo
Anchura	30.0	181058	365	24048	≈ 17s
Voraz con fichas descolocadas	116	803	525	526	≈ 60s
Voraz con Manhattan	66	211	142	143	≈ 33s
A* con fichas descolocadas	30	95920	23489	23530	≈ 16s
A* con Manhattan	30	10439	5101	5102	≈ 16s

Coste del camino

Los mejores han sido la búsqueda en amplitud y ambos A*, seguidos de Voraz con Manhattan y voraz con piezas descolocadas. En términos de eficiencia cuanto menor sea el camino, más rápido será el algoritmo y aquí se demuestra con claridad.
Los algoritmos de búsqueda voraz (basándose solo en las heurísticas) han sido los peores ya que sin acumular costes reales y sin tener en cuenta nodos visitados no encuentran el mejor camino (esto depende de la calidad de la heurística, en todo caso siempre va a ser mejor acumular el coste del camino como hacen los A* en cuanto a eficiencia).

Nodos expandidos

Gana con claridad el algoritmo de búsqueda en amplitud. Aunque no hace ningún cálculo de heurística ha obtenido el mismo tiempo y coste de camino que los algoritmos A*. Entendemos que durante el tiempo que los A* tardan en calcular la heurística, el algoritmo en amplitud sigue expandiendo nodos. De ahí la diferencia de nodos expandidos.

En cuanto a los algoritmos voraces, expande notablemente menos nodos que los anteriores pero sus caminos y tiempos de ejecución son mucho mayores. Esto se debe a que los caminos que siguen no son los más óptimos, a pesar de que utilizan heurísticas válidas.

Tamaño de las colas y máximos

El algoritmo búsqueda en amplitud presenta un tamaño de cola en el momento de finalización menor que el tamaño máximo que ha presentado la cola durante la ejecución, en cambio, el resto tienen el tamaño de cola final similar al tamaño de cola máximo.
Esto significa que en los algoritmos que utilizan heurísticas la cola se ordena cada vez que se expande un nodo. Cuando un nodo no es la solución, se expande y los nodos expandidos se añaden a la cola, después, se ordena. Por lo tanto, cuando encuentra la solución, está en la primera posición de la cola. De ahí que el tamaño máximo y final sea similar en todos estos.

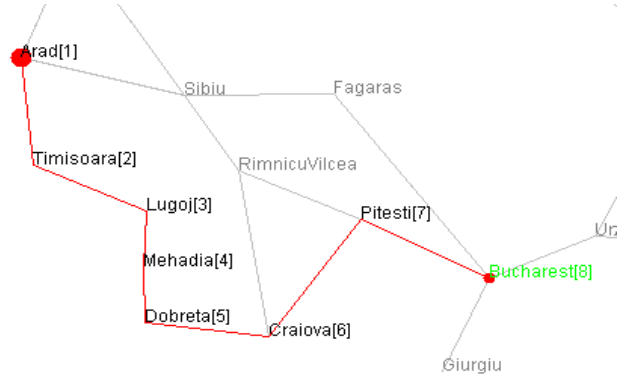
El algoritmo de búsqueda en anchura no se ordena por lo tanto no tiene por qué coincidir el tamaño máximo con el tamaño final, puede encontrar la solución en cualquier punto de la cola.

Tiempo

Los algoritmos de búsqueda voraz basados solo en heurísticas, son los que peor rendimiento han obtenido en tiempo de ejecución. Sin embargo, se puede ver que la heurística de Manhattan es bastante mejor que la de piezas decolocadas.
Los mejores, en cambio, han sido los algoritmos A* que combinan la función heurística con los costes reales.
La búsqueda en amplitud ha demostrado encontrar la solución en tiempos similares a los de los algoritmos A*.

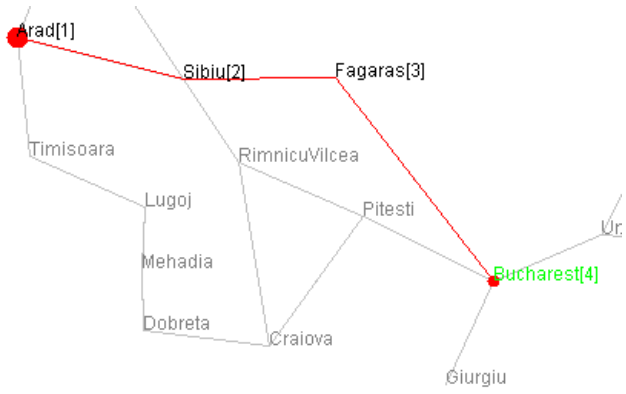
2. AIMA Demo App: Search App

	Coste del camino	Máximo tamaño de cola	Tamaño de cola	Nodos expandidos
Búsqueda en profundidad	733	3	1	10
Búsqueda en anchura	450	5	3	5
Búsqueda con A*	418	6	4	5



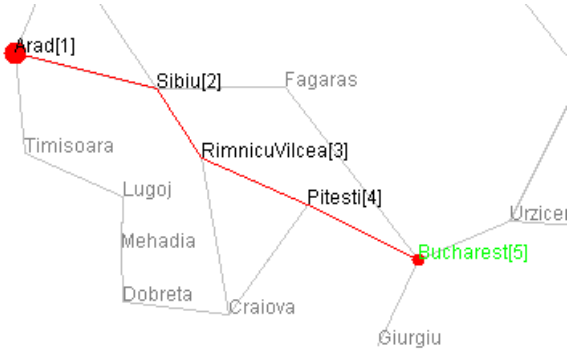
Búsqueda en profundidad

El más costoso sin duda alguna es el algoritmo de búsqueda en profundidad. Como sabemos, este algoritmo no es completo ni óptimo. En este caso ha encontrado una solución y la ha devuelto pero comparando con el resto de algoritmos se ve que no es la mejor solución.
El mejor es el algoritmo de búsqueda con A* porque expande menos o el mismo número de nodos que los otros algoritmos pero su coste de camino es mejor que el de ambos. La combinación de sumar costes, heurística adecuada y el ordenar la cola antes de expandir, es la más eficiente.



Búsqueda en anchura

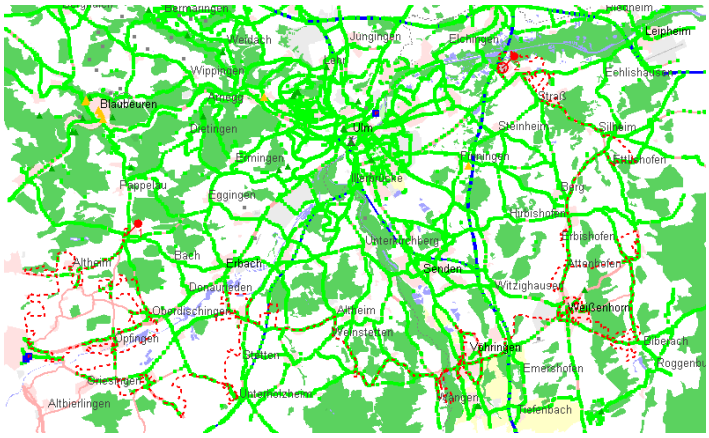
Aunque A* expande el mismo número de nodos que el algoritmo de búsqueda en anchura, sus caminos no tienen el mismo coste.
La explicación de esto es que el algoritmo de búsqueda en anchura asume que cuanta más profundidad tenga el nodo por el que vamos, más coste tiene el camino. Es decir, que llegar a un nodo con profundidad 4 debería ser más costoso que llegar a uno de profundidad 3.
En este caso el nodo de profundidad 4 tiene un coste de llegada menor que el nodo de profundidad 3 por lo que es un caso en el que el algoritmo de anchura no encuentra la solución óptima.



*Búsqueda con A**

3. SearchDemoOsmAgentApp

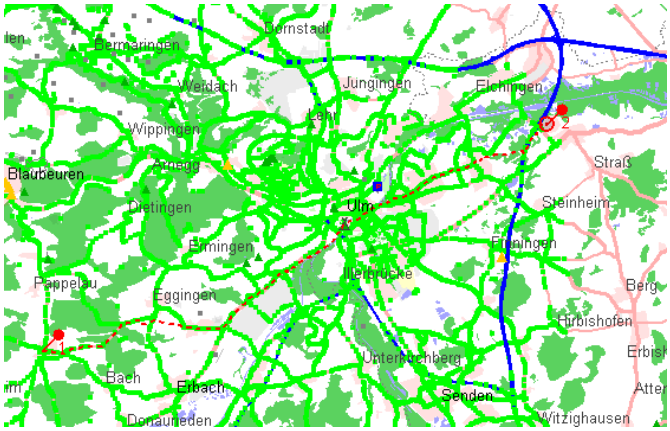
	Coste del camino	Máximo tamaño de cola	Tamaño de cola	Nodos Expandidos
Búsqueda en profundidad	229,413	7081	936	113007
Coste uniforme	26,653	319	153	90908
Búsqueda con A*	26,653	483	335	18947



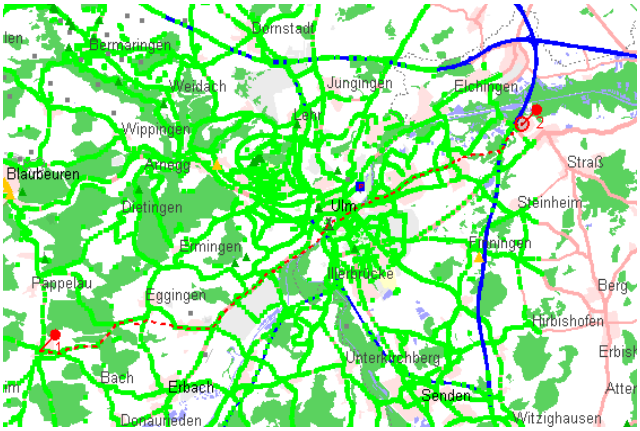
Búsqueda en profundidad

El coste en profundidad (no es óptimo ni completo) ha sido el peor de los 3. No tiene en cuenta el camino recorrido e incluso ha entrado en bucles en algunas partes del recorrido. Esto hace que el algoritmo avance sin control hacia la profundidad del árbol sin tener en cuenta nada y cuando encuentra la solución la devuelve. No es nada eficiente.

En cambio, los otros dos algoritmos han dado soluciones bastante óptimas y parecidas . Realmente ambos han encontrado la mejor ruta pero el A* lo ha hecho más rápido (ha expandido menos nodos).
Esto se debe a que ambos algoritmos utilizan la misma "base". Por así decirlo, el algoritmo A* solo difiere del de coste uniforme en que utiliza la heurística. Esto hace que aunque ambos vayan a encontrar el mejor camino pero el A* lo encuentre antes porque la heurística le da información adicional que le permite "ahorrarse pasos".



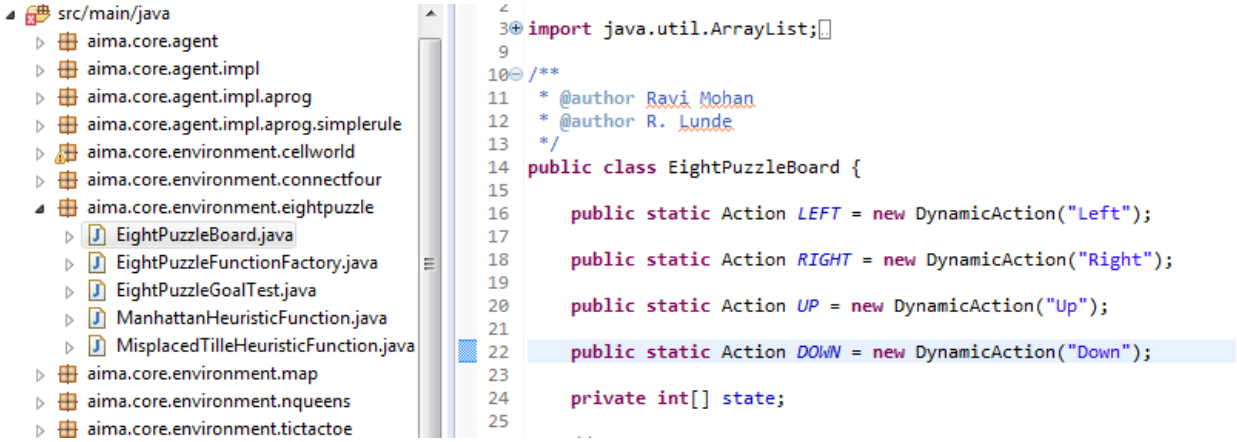
Coste uniforme



*Búsqueda con A**

4. Estado y operadores del Eightpuzzle

Las definiciones de estado y los operadores están contenidos en la clase EighPuzzleBoard.java dentro del paquete aima.core.enviroment.eightpuzzle



Y la heurística del número de fichas mal colocadas está en aima-core/src/main/java/aima.core.enviroment.eightpuzzle/MisplacedTilleHeuristicFunction.java

5. Estado y operadores del mapa de Rumanía

Las definiciones de estado se pueden encontrar en SimplifiedRoadMapOfPartOfRomania dentro del paquete aima.core.enviroment.map. Las definiciones de operadores se encuentran en MoveToAction.java dentro del paquete aima.core.enviroment.map