Actividad 6 – Daniel Marín López
Modelos de Inteligencia Artificial

Índice

Parte 1: Algoritmos de búsqueda evolutivos

Los algoritmos de búsqueda evolutivos son algoritmos cuya forma de representación normalmente se basa en un nodo en forma de árbol generado a partir de un caso inicial que explora las diferentes posibilidades hasta encontrar una solución. Dependiendo de si disponen de información o no, estos se clasifican de la siguiente forma:

- **Búsqueda no informada**
  - Búsqueda primero en anchura
  - Búsqueda primero en profundidad
- **Búsqueda informada**
  - Búsqueda voraz primero el mejor
  - Búsqueda con método A*

Búsqueda primero en anchura

Es el método más sencillo, donde se expande el nodo inicial, después sus hijos y así sucesivamente hasta encontrar la solución. Aquí hay un ejemplo:

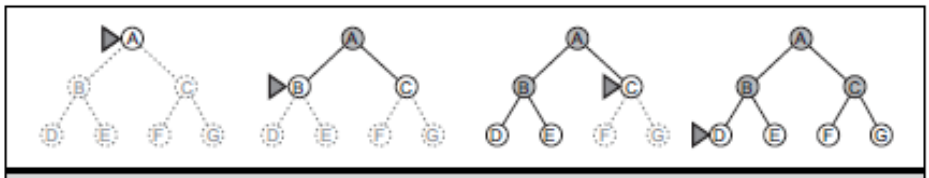


Figura 3.10 Búsqueda primero en anchura sobre un árbol binario sencillo. En cada etapa, el próximo nodo a expandir se indica con una marca.

También hay que tener en cuenta que este método a escala grande no es muy efectivo, ya que el número de nodos sería de $b^{d+1} - b$. Donde b es el n.º de nodos y d sería la profundidad limitada.

Si tenemos en cuenta que cada nivel aumenta en uno tendríamos lo siguiente:

$$b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

Profundidad	Nodos	Tiempo	Memoria
2	1.100	11 segundos	1 megabyte
4	111.100	11 segundos	106 megabytes
6	10^7	19 minutos	10 gigabytes
8	10^9	31 horas	1 terabytes
10	10^{11}	129 días	101 terabytes
12	10^{13}	35 años	10 petabytes
14	10^{15}	3.523 años	1 exabyte

Figura 3.11 Requisitos de tiempo y espacio para la búsqueda primero en anchura. Los números que se muestran suponen un factor de ramificación $b = 10$; 10.000 nodos/segundo; 1.000 bytes/nodo.

Búsqueda primero en profundidad

Este método se parece a la búsqueda en anchura pero con la diferencia que se expande el nodo más profundo a la frontera o límite d. Aquí hay un ejemplo:

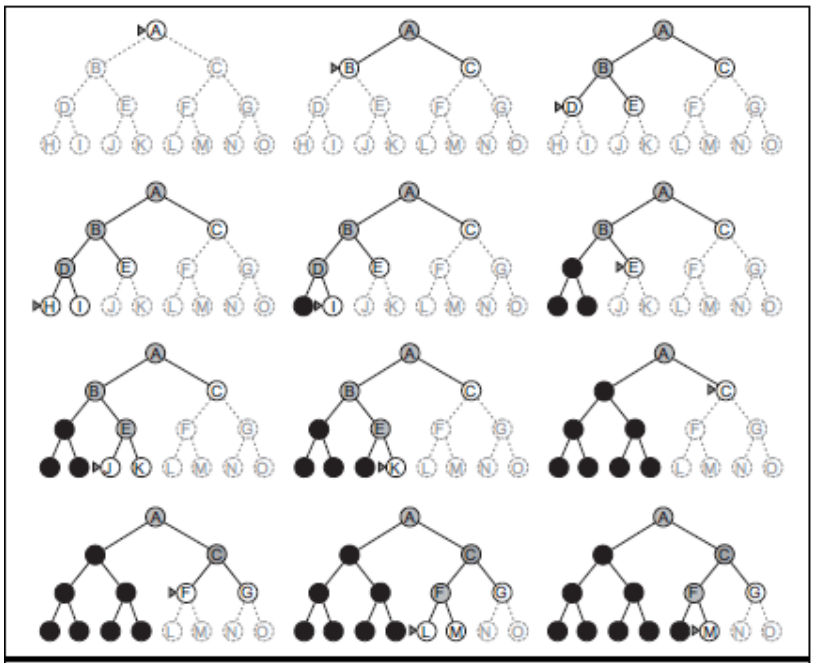


Figura 3.12 Búsqueda primero en profundidad sobre un árbol binario. Los nodos que se han expandido y no tienen descendientes en la frontera se pueden quitar de la memoria; estos se muestran en negro. Los nodos a profundidad 3 se suponen que no tienen sucesores y *M* es el nodo objetivo.

Una forma de implementar este método de manera recursiva en pseudo-código sería de la siguiente forma:

```
función BÚSQUEDA-PROFUNDIDAD-LIMITADA(problema,límite)  devuelve una solución, o fallo/corte
  devolver BPL-RECURSIVO(HACER-NODO(ESTADO-INICIAL[problema]),problema,límite)

función BPL-RECURSIVO(nodo,problema,límite) devuelve una solución, o fallo/corte
  ocurrió un corte ← falso
  si TEST-OBJETIVO[problema](ESTADO[nodo]) entonces devolver SOLUCIÓN(nodo)
  en caso contrario si PROFUNDIDAD[nodo] = límite entonces devolver corte
  en caso contrario para cada sucesor en EXPANDIR(nodo,problema) hacer
    resultado ← BPL-RECURSIVO(sucesor,problema,límite)
    si resultado = corte entonces ocurrió un corte ← verdadero
    en otro caso si resultado ≠ fallo entonces devolver resultado
  si ocurrió un corte? entonces devolver corte en caso contrario devolver fallo
```

Figura 3.13 Implementación recursiva de la búsqueda primero en profundidad.

Hay que tener en cuenta que aquí hay profundidad limitada delimitada por *m*, lo que equivale $b \cdot m + 1$ nodos que se deben de almacenar.

Búsqueda voraz primero el mejor

Este método es un algoritmo de búsqueda informada consiste en buscar el nodo más cercano al objetivo, el cuál probablemente será la solución buscada. Su función heurística es $f(n) = h(n)$.

Un ejemplo sería encontrar una ruta en Rumanía utilizando como función heurística la distancia en línea recta teniendo como nodo objetivo Bucarest.

Arad	366	**Mehadia**	241
Bucarest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figura 4.1 Valores de h_{DLR}. Distancias en línea recta a Bucarest.

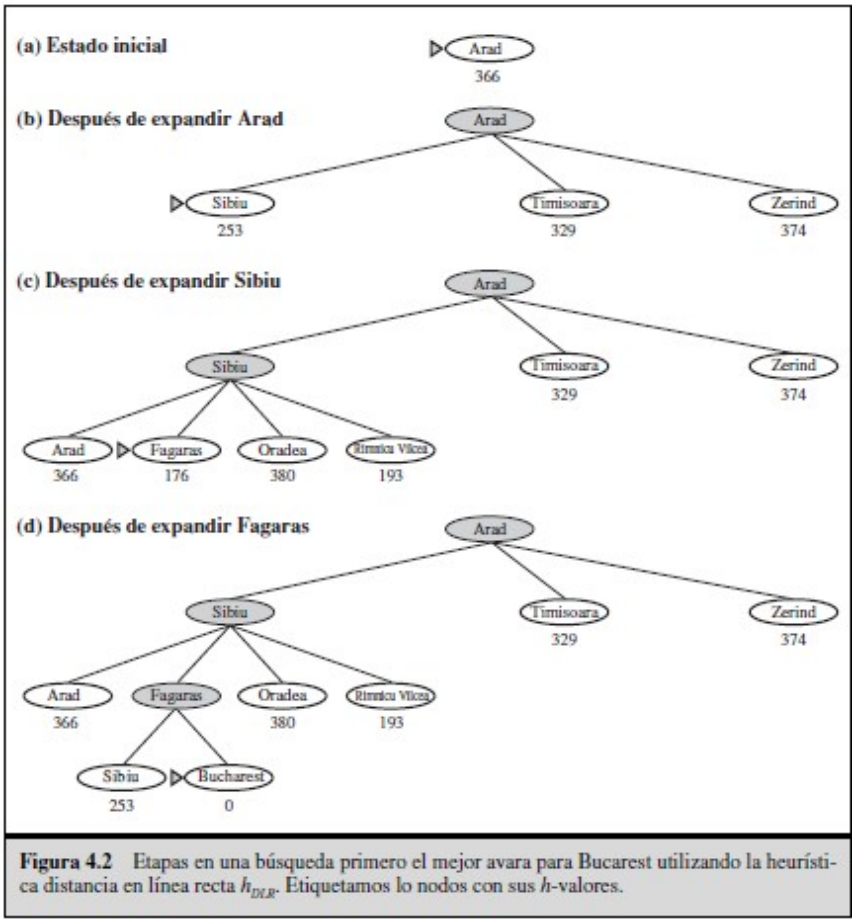


Figura 4.2 Etapas en una búsqueda primero el mejor avara para Bucarest utilizando la heurística distancia en línea recta h_{DLR}. Etiquetamos lo nodos con sus h-valores.

Como podemos apreciar, teniendo en cuenta los n.º de kilómetros buscamos aquella cantidad de kilómetros que sea menor al nodo actual.

Aplicaciones de los algoritmos de búsqueda

Algoritmo	Aplicaciones
Búsqueda primero en anchura	• Resolución de Problemas de Ruta Óptima en Grafos • Análisis e Conexión en Ciencias de Datos y Bioinformática
Búsqueda primero en profundidad	• Solución de problemas de juegos y puzles • Optimización de rutas y navegación autónoma • Generación y análisis de árboles de decisión • Detección de componentes conectados en grafos • Generación de contenido procedimental
Búsqueda voraz primero el mejor	• Procesamiento de Lenguaje Natural (PLN) • Optimización de Rutas y Logística • Sistemas de recomendación

Parte 2: Búsqueda A*

El algoritmo búsqueda A* consiste en un algoritmo que busca minimizar el coste estimado total de la solución del problema. Este evalúa los nodos combinando el costo estimado para alcanzar el nodo (*g(n)*) y el coste para llegar al nodo objetivo (*h(n)*). Esto se representa de la siguiente forma.

$$f(n) = g(n) + h(n)$$

En donde tenemos a *g(n)* que representa el coste del camino desde el nodo inicial al nodo *n*, y *h(n)* sería el coste estimado más barato para llegar desde *n* al nodo objetivo. Lo cual equivale a que *f(n)* representa el coste estimado más barato de la solución a través de *n*.

Algunas aplicaciones donde podemos encontrar el uso de este algoritmo son las siguientes:

- Navegación de Robots y Vehículos Autónomos
- Videojuegos y sistemas de NPCs
- Planificación de Rutas en Sistemas de Información Geográfica (GIS)
- Planificación de Movimientos en Manipuladores Robóticos
- Resolución de Problemas en Tableros y Rompecabezas
- IA para Drones y Exploración de Terrenos Desconocidos
- Sistemas de Logística y Almacenamiento

Ejemplo: Caso del 8-puzle

Dado este 8-puzle:

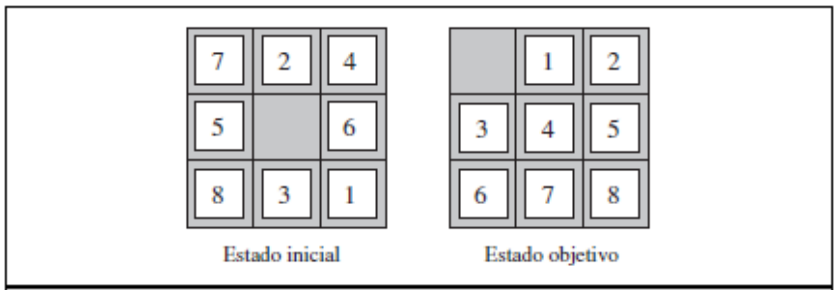
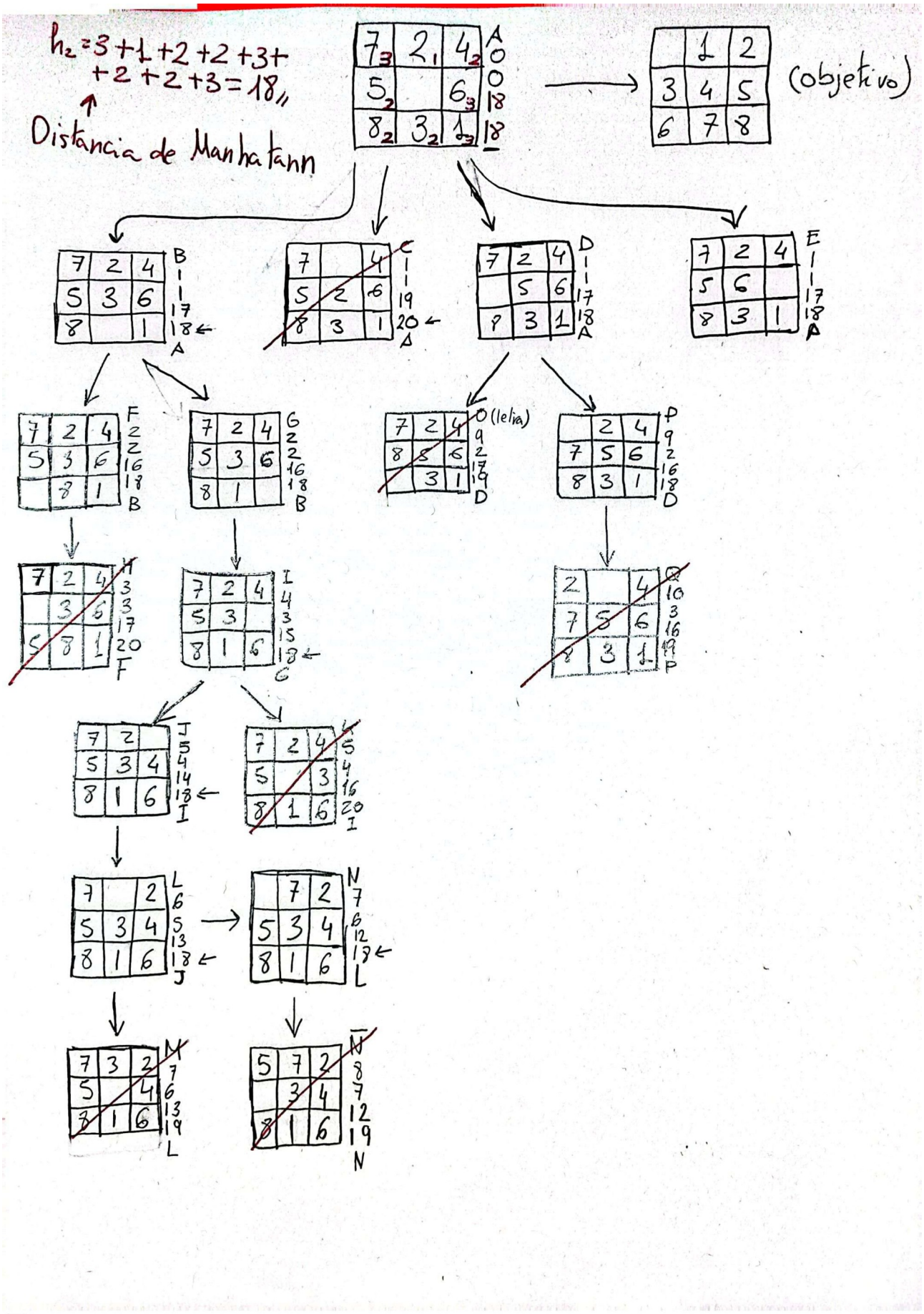


Figura 4.7 Un caso típico del 8-puzle. La solución tiene 26 pasos.

Realiza 10 iteraciones usando el método de búsqueda A*, usa como función heurística la distancia de Manhatann[1] y muestra los siguientes datos por cada iteración:

- ***Nombre de la iteración***
- ***Número de la iteración***
- ***Coste de la iteración***
- ***Valor heurístico de la iteración***
- ***Suma del coste y el valor heurístico***
- ***Nombre del mejor padre***

1. La distancia de Manhatann se refiere a contar el n.º de pasos para colocar una pieza en su sitio (según el nodo objetivo) y sumarlos todos.



Fuentes

- Inteligencia Artificial: Un enfoque moderno (Stuart J. Russell)
- https://chatgpt.com/share/671e326a-93f4-8005-a778-b45f8bda0cc8