Universidad de Granada

Algorítmica

Algoritmos Backtracking, parte 1

Laura Calle Caraballo
Cristina María Garrido López
Germán González Almagro
Javier León Palomares
Antonio Manuel Milán Jiménez

23 de mayo de 2016

Índice

1. Introducción.

El objetivo de esta práctica es el estudio de los algoritmos de tipo *Backtracking*, aplicados particularmente al problema de encontrar un camino desde la entrada de un laberinto hasta su salida.

2. Descripción del problema.

Inicialmente tenemos una matriz de tamaño $n \times n$ que representa un laberinto con o sin solución. Las casillas libres se representan mediante espacios, y los muros mediante X.

Los movimientos permitidos son: norte, sur, este y oeste (no es posible avanzar en diagonal).

3. Resolución.

Se han implementado dos algoritmos *Backtracking*: el primero encuentra un camino (en caso de que exista) y el segundo encuentra el camino más corto posible.

Adicionalmente, se ha realizado un estudio de eficiencia para determinar su viabilidad en términos de tiempo.

4. Algoritmo *Backtracking* sin garantía de optimalidad.

Esta versión del algoritmo construirá el camino probando diferentes alternativas hasta que encuentre una solución completa, momento en el que terminará.

4.1. Pseudocódigo.

A continuación se muestra el pseudocódigo del algoritmo:

```
posActual ⟵ entrada;
camino ⟵ camino ∪ posActual;
function EncontrarCamino(laberinto, posActual);
encontrado ⟵ false;
begin
    if EsSolucion(camino) then
        return true;
    else
        for m ∈ movimientosPosibles and not encontrado do
            h ⟵ GenerarHijo(m);
            if Factible(h) then
                camino ⟵ camino ∪ h;
                encontrado ⟵ EncontrarCamino(laberinto,h);
            end
        end
        if not encontrado then
            camino ⟵ camino − posActual;
        end
    end
    return encontrado;
end
```

5. Algoritmo *Backtracking* con garantía de optimalidad.

Esta versión del algoritmo encontrará un primer camino y continuará la búsqueda de un camino mejor siempre que la longitud de los recorridos que pruebe sea menor que la del mejor camino actual.

5.1. Pseudocódigo.

A continuación se muestra el pseudocódigo del algoritmo:

```
posActual ⟵ entrada;
camino ⟵ camino ∪ posActual;
mejorDistancia ⟵ ∞;
function EncontrarMejorCamino(laberinto, posActual);
encontrado ⟵ false;
begin
    if EsSolucion(camino) then
        if Longitud(camino) < mejorDistancia then
            mejorCamino ⟵ camino;
            camino ⟵ camino − posActual;
            mejorDistancia ⟵ Longitud(camino);
            return true;
        else
            camino ⟵ camino − posActual;
            return false;
        end
    else
        for m ∈ movimientosPosibles and Longitud(camino) < mejorDistancia do
            h ⟵ GenerarHijo(m);
            if Factible(h) then
                camino ⟵ camino ∪ h;
                encontrado ⟵ EncontrarMejorCamino(laberinto,h);
            end
        end
        camino ⟵ camino − posActual;
    end
    return encontrado;
end
```

6. Análisis de eficiencia empírico.

La complejidad algorítmica de estas técnicas hace que el problema sea muy costoso para dimensiones de unas pocas decenas; además, debido al proceso aleatorio de generación de laberintos, los tiempos fluctuarían mucho. Por ello, presentamos una muestra reducida y abarcable de pruebas empíricas.

6.1. Tabla de tiempos de ejecución.

Tamaño	**No óptimo (s)**	**Óptimo (s)**
7	$1{,}57208 \cdot 10^{-11}$	$6{,}21688 \cdot 10^{-11}$
8	$2{,}83452 \cdot 10^{-11}$	$8{,}05098 \cdot 10^{-11}$
9	$7{,}47932 \cdot 10^{-11}$	$2{,}09373 \cdot 10^{-10}$
10	$3{,}22992 \cdot 10^{-10}$	$3{,}85637 \cdot 10^{-10}$
11	$8{,}27965 \cdot 10^{-10}$	$8{,}49879 \cdot 10^{-10}$
12	$3{,}07938 \cdot 10^{-9}$	$2{,}99815 \cdot 10^{-9}$
13	$5{,}35533 \cdot 10^{-9}$	$3{,}31377 \cdot 10^{-9}$
14	$5{,}67689 \cdot 10^{-9}$	$3{,}83898 \cdot 10^{-9}$
15	$8{,}04407 \cdot 10^{-9}$	$5{,}83481 \cdot 10^{-9}$
16	$3{,}08026 \cdot 10^{-8}$	$2{,}84645 \cdot 10^{-8}$
17	$5{,}67211 \cdot 10^{-8}$	$2{,}8893 \cdot 10^{-8}$
18	$5{,}46378 \cdot 10^{-8}$	$2{,}89411 \cdot 10^{-8}$
19	$7{,}24933 \cdot 10^{-8}$	$7{,}94193 \cdot 10^{-8}$
20	$1{,}19869 \cdot 10^{-6}$	$6{,}20915 \cdot 10^{-7}$
21	$2{,}48178 \cdot 10^{-6}$	$1{,}74906 \cdot 10^{-8}$
22	$2{,}53938 \cdot 10^{-6}$	
23	$5{,}41168 \cdot 10^{-6}$	
24	$7{,}09637 \cdot 10^{-6}$	

Figura 1: Tiempos de ejecución de los dos algoritmos.

6.2. Gráficas de tiempos de ejecución.

Podemos observar que ambos algoritmos parecen tener una tendencia de crecimiento exponencial:

6.2.1. Algoritmo no óptimo.

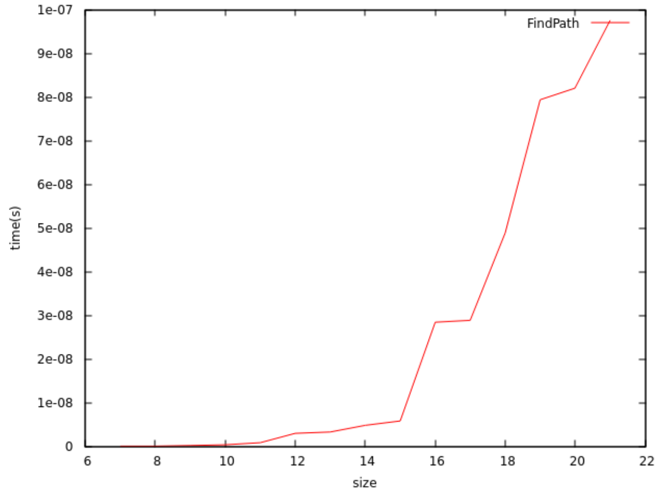


Figura 2: Ejecución del algoritmo que no garantiza la solución óptima. Intel® Core™ i7-5500U CPU @ 2.40GHz.

6.2.2. Algoritmo óptimo.

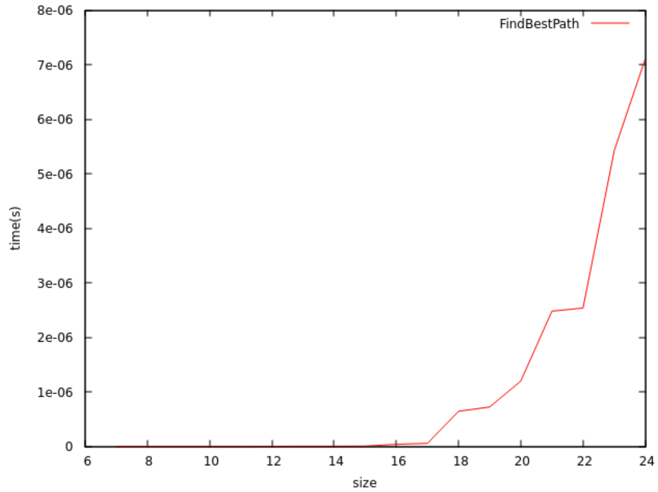


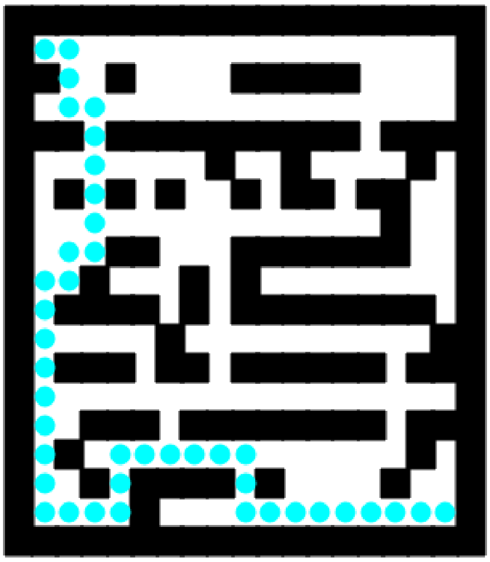
Figura 3: Ejecución del algoritmo que garantiza la solución óptima. Intel® Core™ i7-5500U CPU @ 2.40GHz.

7. Comparativa de calidad de soluciones.

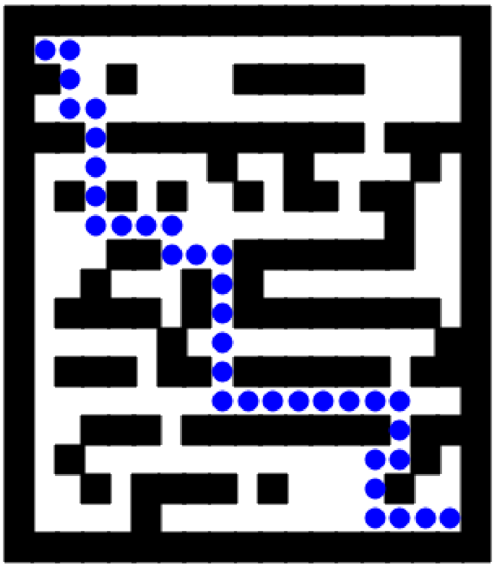
Para mostrar que, efectivamente, el segundo algoritmo realiza una búsqueda más completa y encuentra caminos más cortos en caso de haberlos, compararemos las soluciones obtenidas para un caso concreto.

El laberinto generado es el siguiente:

La solución proporcionada por el algoritmo no óptimo tiene una longitud de 41 y es:



La solución óptima encontrada por el segundo algoritmo, de tamaño 35, es:



8. Conclusión.

La técnica de *Backtracking* permite explorar el espacio de soluciones de forma completa si uno lo desea, pero el coste de hacerlo puede ser demasiado alto. Hemos podido comprobar cómo el algoritmo encontraba soluciones óptimas, lo cual es una ventaja; sin embargo, para tamaños de entrada muy pequeños en comparación con otros algoritmos, *Backtracking* comienza a ser inviable.

En definitiva, deberíamos tener muy en cuenta la dimensión de nuestro problema antes de decidir resolverlo mediante este tipo de algoritmos, ya que la exploración sistemática no suele ser factible en la práctica.