Estructura de datos y Algoritmos.
2º Ingeniería de la Salud

Nombre y Apellidos: Alejandro Domínguez Recio

Práctica 6: Algoritmo de backtracking – Recorridos de un laberinto

Objetivos

El objetivo de esta práctica es aplicar un algoritmo de backtracking que resuelva el problema del laberinto. Para la realización de este se nos proporciona el pseudo-código el cual tenemos que interpretar y así resolver el problema. A continuación se muestra la interpretación en Java de dicho pseudo-código.

```
//      TODO: Implementar el algoritmo de backtracking
//      Comprobamos que se tiene la posibilidad de salir
//      y en caso afirmativo confirmamos con OK
        if (i == L.getExitRow() && j == L.getExitColumn())
            OK = true;
//      Si no podemos salir
        else {
//          Cambiamos el estado de la casilla que no tiene salida
            L.set(i, j, CellState.MARKED);
            int movimientos = 0;
            OK = false;
//          Mientras que tengamos movimientos por hacer
            while (!OK && movimientos < Move.values().length) {
                Move movimient = Move.values()[movimientos];
                int nuevaposicionI = i+ movimient.incX();
                int nuevaposicionJ = j+ movimient.incY();
//              Comprobamos que la nueva posición esta tiene el estado vacio
                if (L.get(nuevaposicionI, nuevaposicionJ) == CellState.EMPTY) {
                    path.add(movimient);
//                  LLamamos recursivamente al método
                    OK = backtracking(new Pair<Integer, Integer>(nuevaposicionI, nuevaposicionJ));
//                  Una vez recorrido ese camino no siendo el correcto disminuimos tamaño
                    if (!OK) {
                        path.remove(path.size()-1);
                    }
                }
                movimientos++;
            }
        }
        return OK;
    }
```

Figura 1.1: Interpretación Java pseudo-código

Configuración

-Características de la máquina en la que se han ejecutado los diferentes algoritmos:

N.º de cores CPU = 1
Frecuencia = 2.20Ghz
HDD = 916Gb
RAM = 8,00 Gb
Versión de Java = JavaSE-13
Sistema operativo de 64 bits, procesador x64
Windows 8.1

-Parámetros con los que se ha ejecutado el algoritmo.

El algoritmo se ha ejecutado con los siguientes archivos de prueba:

- maze0.txt
- maze10.txt
- maze13.txt

A continuación se muestras los resultado mostrados en consola dichos parámetros.

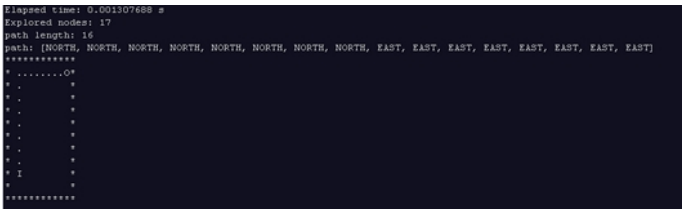

Figura 1.2 : Resultados para maze0.

Figura 1.3 : Resultados para maze10.

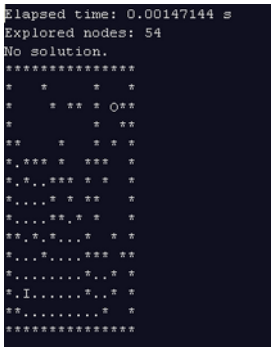

Figura 1.4 : Resultados para maze13.

Resultados

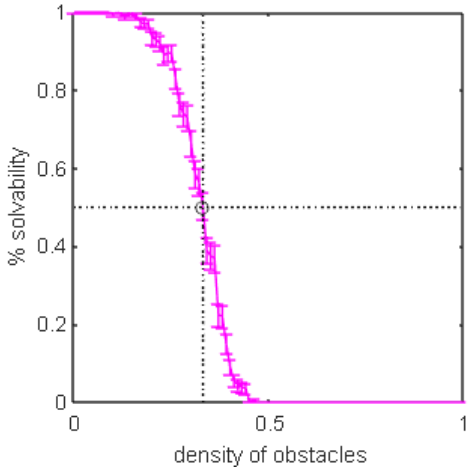

Figura 1.5 : Gráfica generada en Matlab

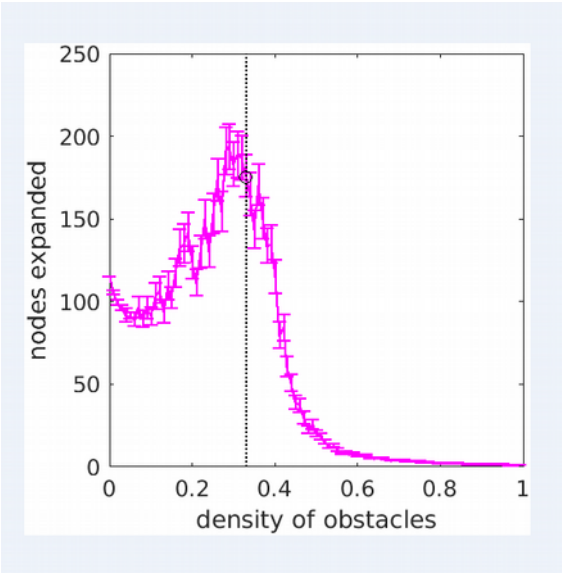

Figura 1.6 : Gráfica 2 generada en Matlab.

Critical solvability point: p = 0.330435

La complejidad del algoritmo la trataremos como una transición de fases ya que nos permite dar respuesta dependiendo de la fase o laberinto. El método de análisis de complejidad visto anteriormente no es aplicable debido a que el algoritmo no posee una complejidad única debido al desconocimiento de las carácteristica de los laberintos a realizar.

PUNTO CRÍTICO

Considerando MAZE como un *constrained satisfaction problem* dependiendo de las limitaciones a la hora de buscar una solución la estructura de estas la dividimos en dos subcategorias:

- *Under-constrained instances :* No existen muchas limitaciónes en la búsqueda de solución en $p < 0.330435$.
- Over-constrained instances : Existen demasiadas limitaciones en la búsqueda de solucíon en $p > 0.330435$.

El punto entre estas dos zonas es el buscado punto critíco de solución.

Critical solvability point: $p = 0.330435$

Mis conclusiones

- La creación de los diferentes caminos y las posibilidades de rechazo según un criterio de restricción con su necesidad de volver hacia atrás para encontrar la solución muestra claramente el funcionamiento de los algoritmos Backtracking.
- El número de movimientos posibles muestra el criterio de restricción.
- El enfoque mostrado corresponde a un problema de decisión.