31-5-2020

Néstor Monzón González - 735418

Andrés Otero García - 757755

PRÁCTICA 2

Ramificación y poda

Algoritmia Básica

1 – Diseño e implementación

El algoritmo de ramificación y poda podría considerarse una variante del backtracking mejorada, ya que, gracias a la poda, cuando se encuentra la primera solución óptima, se asigna su coste como el límite. En la mayoría de los casos, esto permite descartar muchos caminos sin necesidad de expandirlos. Por ejemplo, si se obtiene una solución óptima con un coste de 20, cualquier nodo cuyo coste inicial sea mayor que 20 se descarta (se desactiva). Así, el coste final en tiempo es mucho menor, ya que no es necesario realizar la búsqueda en buena parte de las ramificaciones de un problema dado.

En esta práctica, se ha implementado este algoritmo para aplicarlo a un problema de recorrido de nodos en un espacio bidimensional. Concretamente, contamos con un cierto número de casillas, una posición inicial y una serie de minas repartidas en estas. El objetivo es encontrar uno de los caminos óptimos (más cortos) que pase por todas las minas volviendo a la posición inicial.

Dicho problema se puede extrapolar al problema clásico del mercader ambulante, puesto que se puede definir como un grafo, cuyos nodos representan las minas (y la posición inicial), y cuyos arcos son bidireccionales, y con un coste igual a la distancia L1 entre los nodos que conecta.

A la hora de implementarlo, se decidió utilizar el lenguaje Python, ya que su biblioteca numpy, siendo de alto nivel, está muy optimizada para los cálculos matriciales que cimientan este método (se deben mantener en todo momento una serie de matrices con los costes entre cada mina a visitar).

Desde el punto de vista software, el código se ha dividido en 4 ficheros principales:

* : programa principal. Se encarga, por una parte, de leer y decodificar el fichero de entrada (con la sintaxis descrita en el enunciado), asignando a nuestras estructuras de datos internas los valores del mismo. En segundo lugar, para cada escenario del fichero, llama a la función , que encuentra la solución a un escenario con el método descrito.
* define la clase Escenario con sus correspondientes atributos y métodos. Tipifica el conjunto de parámetros que definen un problema completo: tamaño del espacio, número de minas, posición inicial y posición de cada mina.
* : contiene la función que realiza el algoritmo de poda y ramificación para un escenario, junto con sus distintas funciones auxiliares.
* : define la clase Nodo, con sus datos y métodos.
* : script adicional que utilizamos para generar más ficheros de prueba con escenarios que sirvan de entrada al programa principal.

2 – Pruebas y análisis

Para tener un punto de referencia con el que comparar el algoritmo, se ha implementado uno de fuerza bruta, que simplemente evalúa todos los caminos posibles y toma el mejor. El coste en tiempo de este algoritmo será siempre de (donde es el número de minas), ya que deberá expandir nodos, cada uno generando todos los () restantes, que a su vez generan todos los demás, menos los dos ya expandidos ()...

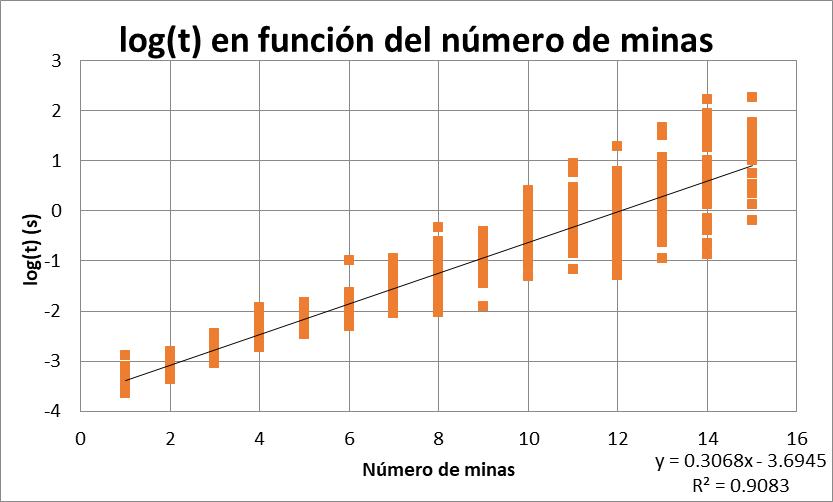
Por otra parte, el algoritmo de ramificación y poda, en el mejor caso, encontrará el camino óptimo en el primer recorrido y descartará todos los demás (expandiendo n nodos, ). En el peor de los casos, no podrá descartar absolutamente ninguno y será idéntico (o incluso ligeramente peor por los cálculos matriciales adicionales) al de fuerza bruta . No obstante, esto no se dará prácticamente nunca: las minas deberían estar todas perfectamente simétricas con respecto a la posición inicial para que se diera el caso en el cual todos los caminos fueran idénticos.

Para obtener datos empíricos de los tiempos, en primer lugar, se ha evaluado el fichero de prueba proporcionado , que contiene 500 escenarios distintos. Se han obtenido las soluciones a cada uno y su tiempo de ejecución. Además, se ha extraído el número de minas de cada uno para relacionarlo con este último valor.

Analizando estos datos, vemos que el número promedio de minas es de 7.97 (y su mediana de 8, es una distribución simétrica). En cambio, los tiempos obtenidos (todos en el rango entre 0.001 y 100 segundos) tienen una media de 2.97 segundos y una mediana de 0.06, lo cual indica una gran diferencia entre unos pocos valores muy grandes relativamente, frente a una gran mayoría de valores en las centésimas.

Esta disparidad refleja que, aunque ciertos escenarios sean muy costosos en tiempo, en los que probablemente haya sido necesario expandir buena parte de los nodos, generalmente se consiguen evitar muchos de estos cálculos. Al representar los datos en un gráfico de dispersión, obtenemos lo siguiente:

A simple vista, se distingue esa asimetría en el tiempo por la que ciertos escenarios han llevado mucho más tiempo que la mayoría. Por otra parte, parece indicar una distribución exponencial. Para comprobarlo, pasamos el eje del tiempo a escala logarítmica (ya que una función exponencial se aproxima a una lineal en escala logarítmica):

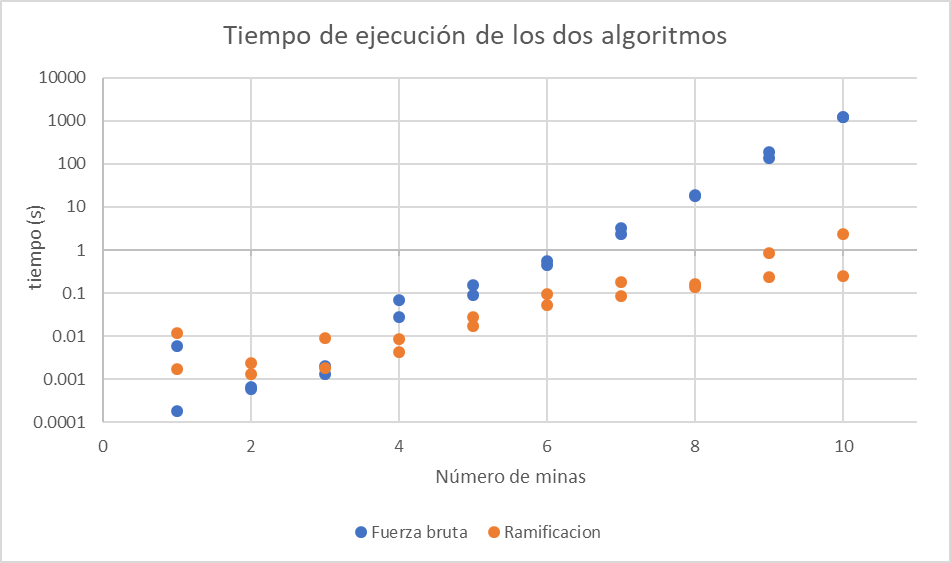


Así, aunque ciertamente haya una gran dispersión (inherente a este algoritmo, ya que, según el escenario, se tienen que expandir un número distinto de nodos sin importar el número de minas del mismo), podemos afirmar que el coste temporal siga una función exponencial en relación al número de minas.

También puede observarse una mayor dispersión a la derecha del gráfico, ya que la desviación (que viene del tiempo de expandir o no un número de los nodos) es proporcional al número de minas.

Finalmente, para asegurar la corrección y eficiencia del algoritmo, se ha realizado la comparación con el algoritmo de fuerza bruta comentado. Hemos comprobado que, para los datos de entrada de (que se ha generado con el script ), ambos devuelven los mismos costes para cada escenario.

Por otra parte, los tiempos de ejecución de cada escenario, en escala logarítmica, se relacionan con el número de minas de la siguiente forma:



Como se puede observar, el algoritmo de ramificación y poda será mejor especialmente con un mayor número de minas. No se han podido comparar números más elevados por una mera cuestión de tiempo con el algoritmo de fuerza bruta, pero ya con 10 es del orden de 1000 veces más lento.