Algoritmos de búsqueda con espacio de memoria limitado

Los métodos de búsqueda por primero el mejor, tales como *A**, no escalan bien a grandes problemas de búsqueda debido a su consumo de memoria. Los algoritmos de búsqueda en haz (*beam search*) reducen el consumo de memoria de la búsqueda por primero el mejor a costa de encontrar soluciones de mayor coste.

1 Búsqueda en haz

Una búsqueda en haz es cualquier técnica de búsqueda en la cual se examinan en paralelo un cierto número de alternativas (el haz), usándose reglas heurísticas para descartar (podar) las alternativas no prometedoras con el fin de mantener el tamaño del haz lo más pequeño posible.

El algoritmo de búsqueda en haz que se pide implementar en este trabajo usa búsqueda en anchura para construir el árbol de búsqueda, pero divide cada nivel del mismo en bloques de a lo sumo B estados, donde B se llama anchura del haz y su valor se establece al inicio de la búsqueda. El número de bloques que se guarda en memoria se limita a uno por nivel. Cuando la búsqueda en haz expande un nivel, genera todos los sucesores de los estados en el nivel actual, los ordena por valor creciente de la función heurística, los divide en bloques de a lo sumo B estados cada uno y extiende el haz almacenando únicamente el primer bloque. La búsqueda en haz termina cuando se *genera* un estado final o cuando se llena la memoria (figura 1).

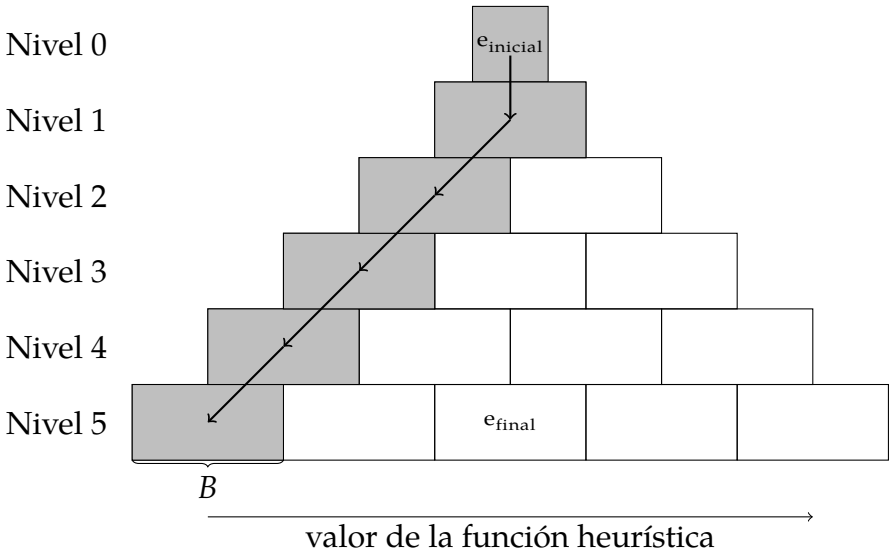

Figura 1: Búsqueda en haz

2 Búsqueda en haz con vuelta atrás

El algoritmo de búsqueda en haz propuesto en la sección 1 no es completo, ya que la poda realizada puede impedir encontrar una solución al problema. Esto no puede resolverse en general variando el tamaño B del haz: para valores pequeños la poda realizada es excesiva,

lo que lleva a no encontrar soluciones o a que las que se encuentren sean costosas; si su valor es grande la memoria se llena rápidamente, lo que impide encontrar una solución.

Una posibilidad para hacer que la búsqueda sea completa es complementar el algoritmo de búsqueda en haz con un proceso de vuelta atrás (*backtracking*) cronológico: cuando la memoria se llena sin encontrar ninguna solución, se elimina de ella el último bloque introducido y se pasa a considerar el siguiente bloque del mismo nivel. Cuando ya se han considerado todos los bloques de ese nivel, se pasa a considerar el siguiente bloque del nivel anterior y así sucesivamente (figura 2).

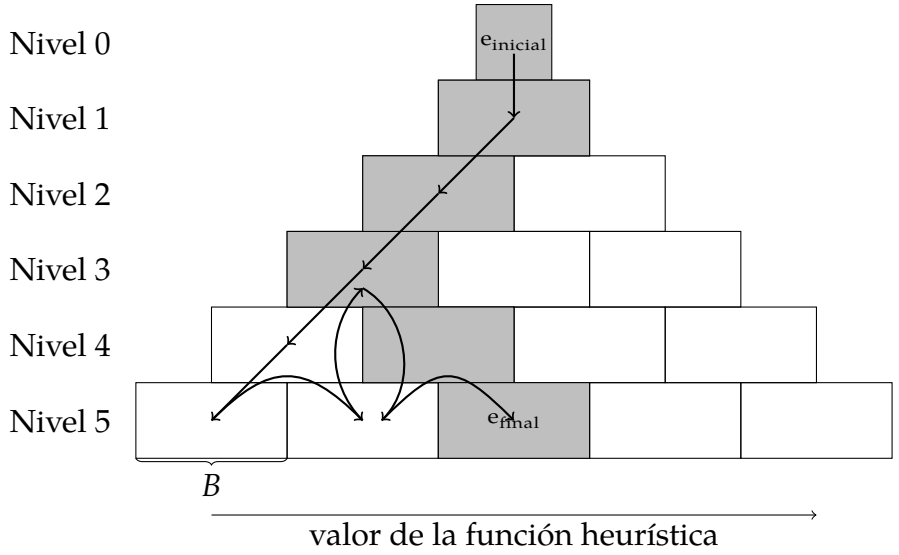

Figura 2: Búsqueda en haz con vuelta atrás cronológica

3 Búsqueda en haz con discrepancias limitadas

La vuelta atrás cronológica revisita primero las decisiones más recientes, es decir, las decisiones cercanas a la parte inferior del árbol de búsqueda. Esto es problemático ya que usualmente los valores de la función heurística son más imprecisos cuanto más lejos se encuentra un estado del estado final, es decir, cuanto más cerca está de la parte superior del árbol de búsqueda. Por tanto, es importante revisitar las decisiones cercanas a la parte superior del árbol de búsqueda más rápidamente.

En [1] se presenta *BULB*, un algoritmo de búsqueda en haz complementado con un proceso de vuelta atrás mediante discrepancias limitadas: se permite seleccionar un número limitado de veces un haz de estados (bloque de a lo sumo B estados) para los que los valores de la función heurística no son los menores posible. Ese número de discrepancias se incrementa de manera iterativa hasta encontrar una solución al problema.

4 El problema del N-puzle

El N-puzle (con $N + 1 = M^2$ para algún M) consta de un tablero $M \times M$ cuyas casillas contienen N piezas numeradas de 1 a N, estando una de las casillas vacía. Los movimientos permitidos son deslizar (en horizontal o vertical, no en diagonal) hacia la casilla vacía alguna de las piezas adyacentes a esa casilla. El problema consiste en obtener el siguiente estado del tablero: la casilla vacía se encuentra en la esquina superior izquierda y, recorriendo el tablero

de izquierda a derecha y de arriba a abajo se encuentran las piezas en orden creciente de numeración.

5 El problema de los N-crepes

El problema de los N-crepes consta de una pila de N objetos (crepes), numerados de 1 a N en cualquier orden. Los movimientos permitidos son invertir el orden de k (con $1 < k \leq N$) elementos de lo alto de la pila. El problema consiste en obtener una pila ordenada de objetos, con el objeto 1 en lo alto de la pila y el objeto N en el fondo de la pila.

6 Objetivos del trabajo

El objetivo de este trabajo es implementar los tres algoritmos de búsqueda en haz presentados y realizar un análisis comparativo mediante una batería de pruebas consistente en resolver instancias aleatorias de los dos problemas descritos.

Para el problema del N-puzle se deben considerar tres colecciones, para $N = 8, 15$ y 24, de 100 instancias aleatorias distintas del problema. Como función heurística se debe usar la distancia Manhattan.

Para el problema de los N-crepes se deben considerar cuatro colecciones, para $N = 30, 40, 50$ y 60, de 100 instancias aleatorias distintas del problema. Como función heurística se debe usar la heurística *gap* descrita en [2].

Dada una de las siete colecciones anteriores de instancias de problemas se debe aplicar cada uno de los tres algoritmos de búsqueda en haz a su resolución, usando el mismo tamaño (adecuado) de memoria para los tres. Debe probarse con distintos tamaños B del haz y calcularse lo siguiente: porcentaje de instancias que el algoritmo ha sido capaz de resolver; coste medio de la solución; número medio de estados generados y de estados almacenados en la memoria; tiempo medio de ejecución.

Consideraciones generales

1. La cantidad de memoria disponible se debe proporcionar a los algoritmos como el número máximo de nodos que puede haber de manera simultánea en el conjunto de nodos explorados y en la frontera.

2. Se considera que el coste de aplicar un operador a un estado es siempre 1. De esta manera, el coste de una solución coincide con su longitud.

3. No estamos interesados en soluciones concretas, sino en la calidad de estas. Por tanto, basta con que los algoritmos proporcionen la longitud (es decir, el coste) de la solución encontrada.

7 Evaluación del trabajo

Para la evaluación del trabajo debe entregarse lo siguiente:

1. Un fichero `análisis_comparativo.pdf` que, para cada algoritmo y para cada colección de instancias de problemas, incluya una tabla que muestre los datos pedidos para cada tamaño B del haz considerado. Debe indicarse también el tamaño de memoria establecido en cada caso, así como las características del ordenador en el que se han realizado las pruebas.

2. Tres ficheros `búsqueda_en_haz.py`, `búsqueda_en_haz_con_vuelta_atrás.py` y `búsqueda_en_haz_con_discrepancias.py` que implementen los tres algoritmos de búsqueda en haz descritos anteriormente.

3. Dos ficheros `N-puzzle.py` y `N-crepes.py` que implementen una función tal que, dados el tamaño del problema, el algoritmo de búsqueda, el tamaño de memoria y el tamaño del haz, permita recrear el análisis realizado para el problema correspondiente. Es decir, el uso de estas dos funciones deben permitir construir las mismas tablas (salvo el dato sobre el tiempo medio de ejecución, que depende del ordenador utilizado) que las presentadas en el fichero `análisis_comparativo.pdf`. Debe documentarse la manera de usar las funciones anteriores.

Además, también se tendrá en cuenta:

- La eficiencia en la implementación y el buen estilo de programación en *Python*.
- La claridad con que se presenten los resultados obtenidos en la memoria y en la defensa.
- Cualquier aportación original que se incluya en el trabajo de manera adicional, aparte de los requisitos mínimos aquí descritos.

8 Bibliografía

[1] Furcy, D; Koenig, S. Limited Discrepancy Beam Search. 2005.

[2] Helmert, M. Landmark Heuristics for the Pancake Problem. 2010.