Práctica 3

Implementación del problema de las jarras

Javier Pellejero Ortega & Zhaoyan Ni

Inteligencia Artificial

Grupo 11

Doble grado Matemáticas e ingeniería informática

1. **Tabla comparativa de los resultados obtenidos con los 8 algoritmos.**

   a) **Capacidad garrafa 1 = 5, capacidad garrafa 2 = 3, cantidad objetivo = 4.**

	Coste del camino	Nodos explorados	Tamaño de la cola	Tamaño máximo de la cola	Tiempo de ejecución en nanosegundos
Búsqueda en Anchura *(TreeSearch)*	6	129	244	244	8702726
Búsqueda en Anchura *(GraphSearch)*	6	11	2	4	351773
Búsqueda en Profundidad *(GraphSearch)*	10	10	9	10	562587
Coste uniforme (*TreeSearch*)	6	476	883	884	7301624
Coste uniforme *(GraphSearch)*	6	13	1	5	297493
Método Voraz *(GraphSearch)*	6	9	6	7	255489
Búsqueda A* (*TreeSearch*)	6	66	123	124	980568
Búsqueda A* *(GraphSearch)*	6	11	1	5	222531

b) Capacidad garrafa 1 = 7, capacidad garrafa 2 = 3, cantidad objetivo = 1.

	Coste del camino	Nodos explorados	Tamaño de la cola	Tamaño máximo de la cola	Tiempo de ejecución en nanosegundos
Búsqueda en Anchura *(TreeSearch)*	4	16	30	30	6165975
Búsqueda en Anchura *(GraphSearch)*	4	7	2	4	371992
Búsqueda en Profundidad *(GraphSearch)*	10	10	10	11	577661
Coste uniforme (*TreeSearch*)	4	51	94	95	5741414
Coste uniforme *(GraphSearch)*	4	9	1	5	864064
Método Voraz *(GraphSearch)*	4	5	4	5	221509
Búsqueda A* (*TreeSearch*)	4	9	17	18	311185
Búsqueda A* *(GraphSearch)*	4	7	3	5	242460

c) Capacidad garrafa 1 = 12, capacidad garrafa 2 = 3, cantidad objetivo = 1.

Para estos datos, el programa no encuentra solución, puesto que no la tiene. Al ser ambas garrafas de capacidades múltiplos de 3, en la garrafa de 12 sólo podremos conseguir las cantidades de 0, 3, 6 ,9 y 12, mientras que en la garrafa de 3 sólo podremos tener 0 ó 3.

2. Análisis de los resultados obtenidos.

En primer lugar, hablemos del único método que no encuentra una solución óptima en ninguno de los dos casos, se trata de la *búsqueda en profundidad* con *GraphSearch*. Esto se debe a que este método de búsqueda no asegura optimalidad pues puede encontrar antes una solución más profunda que la óptima. Además, no supone un ahorro a nivel computacional con respecto a la *búsqueda en anchura* en ninguno de los dos casos contemplados.

El resto de algoritmos, tanto con *TreeSearch*, como con *GraphSearch* encuentran la solución óptima en los dos casos. En particular, la *búsqueda en anchura* encuentra la solución óptima, tanto con *TreeSearch* como *GraphSearch* si el coste de los operadores de búsqueda es uniforme, como es el caso.

En cuanto a la *búsqueda de coste uniforme*, recordemos que considera primero los nodos de menor coste hasta llegar a ellos. Puesto que en este caso el coste de llegar a cada nodo coincide con su profundidad es equivalente a una *búsqueda en anchura* (salvo, tal vez, el orden de la elección de cada nodo en cada nivel de profundidad). Por lo que encuentra la solución óptima ya que *búsqueda en anchura* también lo hace. En caso de duda, añadiremos que este método encuentra la solución óptima si todos los operadores tienen coste mayor o igual que 0, como en este caso.

La *búsqueda A** con *TreeSearch* encuentra la solución óptima porque la heurística que hemos definido es admisible y como además es consistente, también encuentra la óptima con *GraphSeach.* La heurística es admisible puesto que está dividida en tres casos: Devolvemos 0 si hemos llegado al estado objetivo, coincidente con el coste real a la solución, cero; devolvemos 1 cuando podemos llegar al estado objetivo en un paso bien volcando el agua de la garrafa 1 a la 2 o bien viceversa, también coincidente con el coste real; en otros casos, el coste estimado es 2 y siempre es menor o igual que el coste real.

Cabe destacar que no podemos "ganar" vaciando una garrafa y sólo podemos "ganar" llenando una garrafa si la capacidad de alguna de estas es igual a la cantidad objetivo y ese caso se consideraría como hijo del estado inicial y este último nunca es evaluado.

Además, por las razones anteriores, es consistente. Esto garantiza que la solución encontrada por A* tanto con *TreeSearch* como con *GraphSearch* es óptima.

Por último, *voraz* también encuentra la solución óptima en ambos casos, aunque no es lo habitual, en general.

En cuanto al coste de memoria y tiempo de ejecución, cabe la pena destacar la diferencia de memoria usada por el mismo algoritmo con el uso de *TreeSearch* y *GraphSearch*. Este último evita la repetición de estados; es decir, memoriza los estados que ya ha visitado para no volver a ellos en repetidas ocasiones, no así el *TreeSearch*. Aunque esta comprobación tiene un coste en tiempo de ejecución, produce un uso de memoria mucho menor, además de visitar muchos menos nodos, lo que produce a la larga un gran ahorro en tiempo de ejecución.

Una vez señalado el mejor rendimiento de GraphSearch sobre TreeSearch, hablaremos sólo de los algoritmos con uso de este último. Debido a que los datos de los dos casos que hemos analizado no son excesivamente complejos, los resultados son muy similares sea cual sea el algoritmo. Además, la heurística, que en resumidas cuentas se resume en adelantar un paso, tampoco ayuda, pero no hemos encontrado una mejor.

En los casos concretos que analizamos, destacamos el *método voraz* que ha sido más rápido en ambos casos y el que menos nodos ha visitado (*Coste uniforme* y *Búsqueda en* anchura con *GraphSearch* utilizan algo menos de memoria, pero esto es una casualidad, en general usarán más memoria que *voraz*). Sin embargo, es mera casualidad que haya

encontrado la solución (aunque la simplicidad de los casos ayude), así que no podemos fiarnos de este método si queremos encontrar la solución óptima para otros casos.

Como hemos mencionado, la poca información que aporta la heurística hace que no haya grandes diferencias entre los algoritmos informados y los no informados ni en cuanto a nodos visitados, ni en cuanto a memoria utilizada, ni en cuanto a tiempo de ejecución en los casos analizados. Señalaremos que la *búsqueda A** es prácticamente igual de rápida que voraz y supera a *Coste uniforme* y *Búsqueda en anchura* (recordemos que son equivalentes, aunque los datos mostrados puedan no sugerirlo), pero con un coste similar en memoria.

Por último, mencionar que con *TreeSearch* es cuando podemos apreciar una mayor ventaja en todos los aspectos entre los métodos informados (en este caso *A**) y los no informados.