PRÁCTICA 2: Resolución de problemas con búsqueda

Grupo 6

Memoria de la práctica

Javier Guzmán Muñoz **Jorge Villarrubia Elvira**

Parte I: Representación de problemas en espacio de datos.

Ejercicio 1: Representación del puzle de 8

Añadimos las acciones que podemos realizar (las correspondientes a mover el hueco en cualquiera de las cuatro direcciones posibles) a la lista de acciones.

A continuación definimos el resultado de la aplicación de estas acciones, intercambiando el valor del hueco con el de la que casilla que está inmediatamente junto a él en la dirección del movimiento.

Al ejecutar los casos de prueba que aparecen a continuación, se observa como las funciones están correctamente definidas. Al intentar mover el hueco hacia abajo partiendo de una configuración en la que el hueco está en la última fila obtenemos, como era de esperar, un error de ejecución.

Parte II: Ejecución de algoritmos de búsqueda de soluciones

Ejecutamos en primer lugar una búsqueda en anchura (sin control de repetidos) en el espacio de soluciones del problema de las jarras. El algoritmo calcula una solución de 6 pasos. Como el algoritmo es completo siempre encuentra solución, y como es óptima, esta es óptima también.

Ejecutamos una búsqueda en profundidad en el espacio de soluciones y encontramos una solución óptima (hemos tenido suerte, pues el algoritmo de búsqueda en profundidad no es óptimo) pero distinta a la obtenida mediante la búsqueda en anchura.

Ejercicio 2: Búsqueda ciega en 8-puzzle

La primera prueba (búsqueda en anchura sin control de repetidos) no termina porque el problema para la configuración inicial dada no tiene solución y, al no controlar repetidos, el algoritmo entra en bucle y no para nunca.

Volvemos ahora a ejecutar la búsqueda en anchura sin control de repetidos pero con una configuración inicial distinta. El algoritmo termina y encuentra solución en 39.7 microsegundos. El puzle está casi resuelto, solo la última fila esta descolocada, por lo que la búsqueda en anchura obtiene una solución óptima en muy poco tiempo.

Nuevamente ejecutamos el mismo algoritmo, obteniendo un tiempo ahora de 19.9 ms, varios ordenes de magnitud mayor que el anterior, debido a que la configuración inicial ahora está más lejos del estado objetivo y la exploración de las soluciones lleva más tiempo.

La búsqueda en profundidad (con control de repetidos) tarda bastante tiempo en obtener una solución al problema (el tamaño de la solución que encuentra en primer lugar bastante grande,

no tiene por qué ser la óptima, pues el algoritmo no es óptimo). Su ejecución termina al cabo de 11 minutos y 14 segundos.

Ejecutando ahora una búsqueda en anchura con control de repetidos sobre el mismo puzle que las dos búsquedas anteriores, obtenemos un coste de 1.7 ms, muchísimo menor que la búsqueda en profundidad y menor también que si no controlamos los repetidos. Sin embargo, a pesar de su rapidez, este algoritmo tiene un mayor coste en memoria que si no controlásemos los repetidos.

Ejercicio 3: Definir heurísticas para el problema del 8-puzle.

Linear

Nos paseamos por todas las casillas del tablero y, para aquellas que no estén en la posición en la que deberían estar en el estado objetivo, incrementamos un contador de casillas mal colocadas, que se devuelve al final de la función.

Manhattan

Para cada valor en el tablero calculamos su fila y su columna y la fila y la columna que deberían ocupar en el estado objetivo, y calculamos su distancia manhattan con respecto al estado final como la suma del valor absoluto de las diferencias de los índices de las filas y las columnas. Devolvemos la suma de todos estos valores.

Sqrt-Manhattan

Calculamos la raíz cuadrada del resultado de aplicar la función anterior.

Max_heuristic

Devolvemos el máximo de las funciones manhattan y linear aplicadas sobre el nodo que recibe esta función como argumento.

Ejercicio 4: Búsquedas con heurística

Al ejecutar la búsqueda A* con la heurística que viene definida en la clase del problema del 8 puzle por defecto (devolver siempre 1), se obtiene un tiempo de ejecución elevado (8.87 ms). Esto es debido a que la heurística es muy poco informada y la aplicación del algoritmo es equivalente a ejecutar una búsqueda ciega sobre el árbol de soluciones del problema. La solución obtenida tiene 8 pasos hasta llegar al estado óptimo.

Al aplicar la búsqueda A* con la heurística basada en la distancia manhattan obtenemos una solución en 8 pasos idéntica a la obtenida con la heurística poco informada que venía en la definición del problema. Sin embrago, el tiempo de ejecución del algoritmo ha sido de 222 microsegundos, muchísimo menor que el anterior, debido a un mayor grado de información en la heurística.

Al aplicar ahora la búsqueda A* con la heurística lineal volvemos a obtener la misma solución pero esta vez en 209 microsegundos.

La distancia total lineal para este ejemplo es de 3 unidades, mientras que la distancia manhattan es de 4, valores tan parecidos que proporcionan un grado de información bastante similar para ambas heurísticas, teniendo por tanto un tiempo mayor en la manhattan, debido a que es más costoso calcular el valor de la función heurística en este caso.

Al ejecutar el algoritmo con la heurística que toma el máximo de las dos anteriores, obtenemos un resultado peor que tomando ambas por separado (254 microsegundos) debido a que cada

vez tiene que calcular ambas heurísticas y como su grado de información era bastante similar las diferencias de tiempos al aplicar una u otra no son considerables con respecto al coste de llamar a cada una de ellas todas las veces.

Probamos ahora con 3 puzles distintos la búsqueda A* con las diferentes heurísticas.

Veamos primero que devuelven las funciones heurísticas para las configuraciones iniciales de cada puzle

	Lineal	Manhattan	Sqrt_Manhattan	Max_heuristic
Puzle 1	3	4	2	4
Puzle 2	2	2	1.41	2
Puzle 3	2	6	2.45	6

Heurística distancia lineal:

Puzle 1: 206 microsegundos, la solución óptima se alcanza en 8

Puzle 2: 26.7 microsegundos ya que la heurística está completamente informada (su valor coincide con el coste de llegar a la solución óptima (2 movimientos)), de ahí un tiempo de ejecución tan bajo.

Puzle 3: 2.31 ms. Es el que más tarda ya que la distancia lineal es 3 y los pasos necesarios para llegar a la solución son 14, es decir, para este caso concreto la heurística está menos informada que para los otros dos.

Heurística manhattan

Puzle 1: 220 microsegundos.

Puzle 2: 40.3 microsegundos. Aquí, al igual que con la lineal, tenemos que la heurística está completamente informada aunque tarda más que la lineal porque se necesita más tiempo para calcular los valores de la función.

Puzle 3: 1.24 milisegundos.

Para el puzle 3 obtenemos un mejor resultado que con la lineal, porque la heurística manhattan en el estado inicial para este caso está más informada que la lineal (6 frente a 2) para una solución de 14 movimientos.

Heurística Sqrt_manhattan

Puzle 1: 1.22 milisegundos. Tardará en calcular la raíz cuadrada cada vez que calcule la heurística. Además son valores más bajos que la distancia manhattan, por lo que tardará más en encontrar la solución (heurística menos informada).

Puzle 2: 41.5 microsegundos. Como aquí el estado inicial está cerca de la solución óptima no se aprecia una gran variación en el tiempo con respecto a los casos anteriores, aunque aquí también la heurística sigue siendo peor.

Puzle 3: 24.3 milisegundos. Aquí el empeoramiento de rebajar el valor de una heurística que ya era consistente tiene unas consecuencias más marcadas, con un aumento de hasta 20 veces en el tiempo de ejecución con respecto a la distancia manhattan, además del sobrecoste de recalcular la raíz cuadrada en cada paso de la búsqueda.

Heurística del máximo:

Puzle 1: 250 microsegundos: peor que las dos de las cuales calcula el máximo ya que lados heurísticas devuelven valores bastante similares.

La distancia máxima tiene sentido cuando las dos heurística tienen casos de nodos en los que son considerablemente mejores que las otras, ya que con esto nos aseguramos en cada nodo siempre tomar el mejor valor entre los dos de las otras dos heurísticas, asumiendo el sobrecoste de calcular dos valores en lugar de 1.

Puzle 2: 46.9 microsegundos, De nuevo como las dos heurística valen lo mismo no tiene sentido aplicar esta (doble coste), pues estamos en la situación opuesta a la ideal descrita anteriormente.

Puzle 3: 1.38 ms. Mejora a la lineal pero no a la manhattan, indicándonos esto que para la mayoría de los nodos la mejor heurística ha sido la manhattan y casi nunca se ha tenido que decantar por la lineal. Tenemos entonces que tampoco estamos en el caso óptimo descrito para la aplicación de esta heurística.

Parte III: Calcular estadísticas sobre la ejecución de los algoritmos para la resolución de problemas del 8-puzle.

Ejecutamos la búsqueda A* con la clase de problema normal, tardando 49.2 microsegundos, mientras que si la ejecutamos sobre la clase modificada que informaba también de la existencia de la solución el coste es ligeramente superior (52 microsegundos). La diferencia de coste es mínima ya que la solución se alcanza en un solo movimiento.

Al ejecutar la función que da detalles de la búsqueda obtenemos que el número de nodos analizados es dos (0 y 8) y que la longitud tiene solución 1 (solo hace falta hacer un movimiento del hueco a la derecha para llegar a la solución óptima.

Ejercicio 5: Tabla de comparación de estadísticas de los distintos algoritmos.

En primer lugar, observamos que el peor de los algoritmos propuestos es la búsqueda en profundidad. Sería bueno en casos muy concretos donde la solución esté justo en las primeras ramas exploradas, y parece ser que aquí no se da tal caso. Siempre es peor que la búsqueda en anchura, no solo en tiempo, sino que además nos saca soluciones con una longitud desproporcionada, muy lejos de las óptimas.

La búsqueda en coste uniforme para los casos propuestos nos da también una solución óptima, pero sigue siendo peor en tiempo de ejecución que la búsqueda en anchura.

Al añadir heurísticas a las funciones de búsqueda, la cosa mejora bastante. Como era de esperar, el mejor algoritmo es el A* cuando hace uso de la función heurística basada en la distancia manhattan, ya que este algoritmo es más eficiente que el de primero el mejor y esta es la heurística más informada de las que estamos probando. Además, en el algoritmo A* la consistencia de la heurística lineal y manhattan nos garantiza optimalidad en las soluciones obtenidas puesto que estamos utilizando su versión con control, de repetidos.

Ejercicio 6: Definir una nueva heurística más informada que las anteriores

La heurística elegida está basada en buscar costes de la solución de subproblemas del problema dado. La idea la hemos obtenido del libro de la asignatura *Inteligencia Artificial: Un enfoque moderno*, de Stuart Russell y Peter Norving. Así, para el problema del 8-puzle consideramos el subproblema suyo consistente en que no todas las casillas tienen que estar colocadas en sus respectivas posiciones, sino que consideraremos como estado final todo aquel en el que las cuatro primeras casillas (casillas 1-4) estén correctamente colocadas. Tomamos también la idea

del modelo de base de datos, y precalculamos todas las maneras de llegar a estos estados objetivos del subproblema.

1	2	3
4	*	*
*	*	H

En primer lugar calculamos todos los estados objetivo posibles con las cuatro primeras casillas en su posición correcta y el hueco en la última posición, es decir, uno para cada permutación de las casillas 5, 6, 7 y 8, las cuales pueden aparecer en cualquier orden.

Una vez obtenidas estas configuraciones del tablero, vamos a intentar almacenar el coste de llegar a ellas desde el mayor número de estados iniciales posibles. Para esto realizamos la exploración en sentido inverso, es decir, partimos de cada uno de los estados objetivo del subproblema calculados previamente y vamos expandiendo el árbol en sentido inverso, es decir, para cada estado resultante de realizar movimientos de fichas desde la configuración inicial almacenamos en un mapa dicho estado y el coste de llegar hasta él, que será el valor de esta heurística para el nodo al que hemos llegado. Como hay 9! configuraciones iniciales posibles, es impensable tener precalculadas las maneras de llegar a cada nodo objetivo para cada una de ellas, por lo que ponemos un límite a la profundidad en el árbol de exploración y lo fijamos en 15, es decir, que solo vamos a tener precalculadas las soluciones para nodos en los que podamos llegar a un estado con las cuatro primeras casillas colocadas en un máximo de 15 movimientos. Esta exploración la hacemos a partir de una búsqueda en anchura limitando la profundidad a 15. Almacenamos cada configuración avanzada y el número de pasos para llegar a ella (que en realidad serían el número de movimientos para llegar desde ese estado hasta un estado objetivo del problema relajado) en un mapa, y así podemos obtener el valor de la heurística para una configuración del tablero dada accediendo únicamente a estos datos almacenados y que sólo tendríamos que calcular una vez.

Esta heurística es admisible, pues toda solución del problema es también solución del sub problema y además, es más informada que la distancia manhattan.

Como se dará el caso de que nos encontremos con nodos para los cuales no tengamos preprocesado el valor de esta heurística, recurrimos a la técnica de definir como heurística el máximo de otras dadas. Si para un nodo dado no tenemos su valor en el mapa, devolvemos la distancia manhattan para dicho nodo, y en caso contrario devolvemos el valor máximo entre la distancia manhattan para dicho nodo y el valor obtenido previamente y que está guardado en el mapa.

Como esta heurística que habíamos guardado en el mapa es mejor que la manhattan, podemos decir que el valor que estamos devolviendo será igual o mejor que el devuelto por la distancia manhattan, por lo que ya tenemos una heurística que mejora a las definidas previamente.

Sin embargo, comprobamos que al aplicar A* con esta heurística para los cuatro puzles del ejercicio 5 obtenemos peores resultado en cuanto a nodos visitados y tiempo de ejecución que al aplicar el mismo algoritmo con la heurística de la distancia manhattan, lo cual puede ser debido al hecho de que una heurística sea más informada no nos garantiza en general que el algoritmo de búsqueda vaya siempre a obtener las soluciones en un tiempo menor, y se ha dado que para estos cuatro casos con las cotas dadas por la distancia manhattan exploramos antes las ramas con las que llegamos a la solución.

Problema de los misioneros

Para obtener el coste en memoria de los distintos algoritmos a probar, vamos a utilizar la función que nos proporcionaba estadísticas de la ejecución de los mismos definida para el 8 puzle. Recogemos las estadísticas resultantes de la ejecución de los algoritmos en la siguiente tabla. Los algoritmos de primero el mejor y A* han sido probados usando la heurística trivial definida en la clase ProblemaMisioneros.

Algoritmo	Tiempo	Longitud	Nodos analizados
Anchura sin controlar repetidos	148 ms	11	11878
Anchura controlando repetidos	998 µs	11	15
Profundidad sin controlar repetidos	Lo tenemos que parar antes de que termine		
Profundidad controlando repetidos	997 µs	11	12
Búsqueda de coste uniforme	998 µs	11	15
Búsqueda primero el mejor	0 ns	11	13
Búsqueda A*	0 ns	11	15

Optimalidad de las soluciones obtenidas

Analizando los resultados de la tabla podemos concluir que todos los algoritmos nos proporcionan una solución óptima (de longitud 11). Sabemos que todas son óptimas porque tienen la misma longitud que la obtenida, por ejemplo, con el algoritmo de búsqueda en anchura, que sabemos que es óptimo.

El algoritmo de búsqueda en profundidad no es óptimo, pero en este caso tenemos suerte y la solución que nos devuelve sí que lo es. Además, obtenemos una solución en 11 movimientos analizando solo 12 nodos, lo que nos indica que la solución está en la primera rama explorada.

Idénticamente ocurre con el algoritmo de búsqueda heurística primero el mejor, que al estar basado en una búsqueda voraz no es ni completo ni óptimo, pero nuevamente para este caso la solución obtenida sí que es óptima.

Con A*, al ser la heurística h (devolver siempre 1) admisible y consistente, sí que tenemos garantizada la optimalidad de la solución.

Por último decir que hablamos de "una solución óptima" y no de "la solución óptima" porque en las distintas ejecuciones hemos obtenido al menos dos soluciones distintas de igual profundidad (la que devuelve la búsqueda en anchura y la que devuelven los demás), lo que nos indica que la solución óptima no es única.

Coste en memoria

El único dato que tenemos que pueda darnos una idea del coste en memoria de la aplicación de los algoritmos de búsqueda de soluciones es el número de nodos del árbol de exploración analizados por cada uno.

Para todos los algoritmos aplicados con control de repetidos (los del tipo graph) observamos que el número de nodos analizados oscila entre 12 y 15 para obtener soluciones de profundidad 11, lo que indica que gracias al control de repetidos nos ahorramos visitar muchos nodos que no nos llevan a la solución del problema. Este hecho se pone de manifiesto cuando observamos la cifra de nodos analizados para la búsqueda en anchura sin control de repetidos (11878 nodos), lo que nos hace pensar que las búsquedas con control de repetidos tendrán un sobrecoste en memoria considerable para ser capaces de reducir tanto el número de nodos analizados. Este sobrecoste de almacenar los nodos visitados en memoria podría verse compensado con una

mejora en el coste en tiempo (998 µs con control de repetidos frente a 148 ms), pues nos ahorramos visitar una cifra cercana a los 12000 nodos.

Coste en tiempo

En este aspecto, comentar que, exceptuando la búsqueda en profundidad sin control de repetidos (cuya ejecución tenemos que interrumpir tras un período de tiempo prudencial), todas obtienen una solución en poco tiempo, aunque podemos observar ciertas diferencias.

El mayor tiempo se corresponde con la búsqueda en anchura sin control de repetidos, consecuencia directa del mayor número de nodos analizados con respecto a las otras búsquedas.

Los menores tiempos (0 ns, indetectables por el medidor) se obtienen para las búsquedas A* y primero el mejor con la heurística trivial (muy poco informada), lo que nos hace pensar que con cualquier heurística algo informada podemos obtener soluciones con gran rapidez.

Ejercicio opcional

Hemos definido para esta parte de la práctica las dos heurísticas que viene definidas en las transparencias de la asignatura y que coinciden con las que hemos encontrado consultando diversas fuentes en la red.

La primera heurística (h1_misioneros) consiste en relajar el problema eliminando la restricción de que solo haya una barca y siempre tenga que volver alguien desde la orilla derecha para transportar a más personas. Así, suponemos que en cada viaje movemos a un misionero y a un caníbal y que nunca nadie tiene que realizar el trayecto de vuelta (pues hay barcas infinitas), evitando así estado de peligro pues siempre mantenemos el mismo número de caníbales y misioneros en cada orilla del río. Por tanto, el número de viajes a realizar será la mitad del número total de personas (caníbales + misioneros) que queden en la orilla izquierda del río.

La segunda heurística (h2_misioneros) aparece tras eliminar la restricción de que los caníbales se comen a los misioneros cuando les superan en número. Así, en cada viaje de ida transportamos a una persona (en realidad llevamos a dos pero una tiene que volver con la barca, por lo que esto equivale a hacer viajes solo de ida con una persona), y por cada persona transportada tenemos que hacer dos viajes (ida y vuelta). Consideramos que para transportar a cada persona su viaje comienza en la orilla derecha cuando la barca vuelve y acaba cuando vuelve a llegar a dicha orilla dejando allí a la persona transportada. Así, los movimientos acabarán con la barca en la orilla derecha. Sin embargo, si la barca comienza en la orilla derecha, podemos ahorrarnos un viaje inicial de vuelta para la primera persona transportada, por lo que restamos 1 al número de viajes a realizar si se diese esta situación.

$$h_2 = 2(numCanibales + numMisioneros) - orilla$$

$$orilla = 1 \; si \; la \; barca \; comienza \; en \; la \; orilla \; izquierda$$

$$orilla = 0 \; en \; caso \; contrario$$

Ambas heurísticas son admisibles y consistentes, de ahí que A* devuelva para ambas solución óptima.

Además lo hace en un tiempo imperceptible para el medidor de tiempos (devuelve 0 ns) y con ambas heurística el número de nodos analizados es de 14 para soluciones de profundidad 11, por lo que podemos decir que encuentra las soluciones óptimas con holgada rapidez.