BHFFA*: Un nuevo algoritmo admisible de búsqueda bidireccional

Carlos Linares López Asunción Gómez-Pérez
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
28660 - Boadilla del Monte
Madrid - España
e-mail: {clinares,asun}@delicias.dia.fi.upm.es

Resumen

A pesar de que inicialmente hubo un gran interés en los algoritmos de búsqueda bidireccionales, muy pronto se pensó que garantizar la optimalidad de las soluciones encontradas de este modo era muy complicado, y por ello se desestimó esta línea de investigación. En este artículo se muestra, sin embargo, que es posible superar los principales inconvenientes de la búsqueda bidireccional y desarrollar un nuevo algoritmo admisible, con una heurística consistente, y en términos muy sencillos. Además, a diferencia de otras implementaciones bidireccionales, la que se muestra aquí puede resultar en reducciones del tiempo necesario y de la memoria consumida de hasta el 99%, y siempre superior a su implementación unidireccional. Para constatarlo, se han estudiado dos dominios radicalmente diferentes: el grafo del Metro de Madrid y el juego del N-'Puzle'.

1 Introducción

Desde la publicación del algoritmo A* [Hart, 1968, Nilsson, 1987] se han sucedido diversos intentos para mejorar su eficiencia. De hecho, muy tempranamente, se observó que era posible disminuir el esfuerzo computacional desarrollando dos búsquedas al mismo tiempo. Una desde el nodo inicial, s, y otra desde el nodo final o meta, t. El objetivo era que ambas coincidiesen en un nodo común, n. Si este nodo existía, la solución resultaba, de la concatenación del camino que parte del extremo inicial, hasta el nodo de coincidencia, P_{s-n}, y desde éste hasta el nodo meta, P_{n-t} [Pohl, 1969]. Además, los algoritmos de búsqueda bidireccionales parecen fácilmente implementables en arquitecturas paralelas, lo que reduciría considerablemente sus tiempos de respuesta.

En ausencia de información heurística, los algoritmos de *fuerza bruta* pueden beneficiarse sensiblemente con la utilización de la búsqueda bidireccional [Pohl, 1987]. Sin embargo, el caso de los algoritmos de *búsqueda heurística* es bien distinto. Al contrario que los algoritmos de fuerza bruta, que exploran exhaustivamente y de manera ordenada todo el espacio de estados, los algoritmos de búsqueda heurística dirigen la búsqueda gracias a la utilización de una *función heurística*, $h(n)$, que es una estimación del esfuerzo necesario para alcanzar el estado meta a partir del estado n. Si la función heurística es exacta, esto es, si la estimación del coste para alcanzar el nodo final, $h(n)$, es igual al coste del camino óptimo hasta dicho nodo, $h^*(n)$, entonces el algoritmo A* encontrará la solución en el menor número de pasos [Pearl, 1984], por lo que ningún algoritmo bidireccional mejoraría sus resultados. Sin embargo, en el caso frecuente de que $h(n)$ sea diferente de $h^*(n)$, los algoritmos bidireccionales dirigirán la búsqueda en direcciones incorrectas, ocasionando imprecisiones. Estas imprecisiones complican la búsqueda bidireccional puesto que puede ser difícil garantizar que ambas búsquedas coincidan en un nodo intermedio y porque aunque coincidiesen, garantizar la optimalidad de las soluciones encontradas no es trivial. De hecho, algunos experimentos han demostrado que la búsqueda heurística

bidireccional consume, frecuentemente, el doble de los recursos de una sola búsqueda unidireccional [Politowski, 1984].

El algoritmo BHPA es el primer intento de implementar el A* bidireccionalmente y se debe a Ira Pohl [Pohl, 1969]. Consiste en aplicar dos búsquedas simultáneamente: una, desde el nodo inicial, buscando el mejor camino hasta el nodo meta, P^*_{s-t}; la otra, desde el nodo final, buscando el camino óptimo hasta el nodo inicial, P^*_{t-s}. Desafortunadamente, no solo ocurre que el algoritmo BHPA no es un algoritmo admisible, sino que además es difícil garantizar que habrá coincidencia entre ambas búsquedas, dado que ambas se desarrollan independientemente. Para solucionar este problema, Politowski y colaboradores[Politowski, 1984] presentaron el algoritmo de *redireccionamiento al d-nodo*, que consiste, simplemente, en elegir un nodo de la búsqueda realizada en sentido opuesto para dirigir la búsqueda hacia él. Evidentemente, este algoritmo garantiza la coincidencia entre ambas búsquedas pero, con frecuencia, las soluciones alcanzadas no serán óptimas. Esto se debe a que el nodo arbitrariamente elegido (denominado *nodo d*), puede no formar parte de la solución óptima.

Más tarde, Champeaux y colaboradores [Champeaux, 1977] propusieron el algoritmo BHFFA, que se presentará en la sección siguiente. Desafortunadamente, aunque este algoritmo garantiza, razonablemente, la coincidencia entre ambas búsquedas, no es admisible.

Por todos estos motivos, la comunidad científica desestimó la investigación de los algoritmos bidireccionales, y así lo testimonia la pobre bibliografía al respecto. Mientras que los algoritmos unidireccionales han sido objeto de constantes mejoras, han existido muy pocas aportaciones a la búsqueda bidireccional.

En 1989, Kwa propuso el algoritmo BS* [Kwa, 1989], una versión admisible del algoritmo BHPA que, sin embargo, tenía un rendimiento inferior al del A*.

Recientemente, Nelson y Dillenburg [Dillenburg, 1994] han propuesto los algoritmos *de perímetro* como una nueva forma de implementar la búsqueda bidireccional. Dado que los inconvenientes mencionados anteriormente se derivan del hecho de ejecutar ambas búsquedas simultáneamente, ellos han propuesto desarrollar: primero, una búsqueda alrededor del nodo final, creando un conjunto de nodos denominado *perímetro* cuya distancia y camino óptimo hasta la meta, P^*_{n-t}, es conocido; segundo, una búsqueda unidireccional desde el nodo inicial hacia los nodos del perímetro.

Más tarde, Manzini propuso el algoritmo BIDA* [Manzini, 1995]. BIDA* es un algoritmo de perímetro que utiliza el algoritmo IDA* en la búsqueda a partir del nodo inicial. La diferencia de este algoritmo con el de Nelson y Dillemburg consiste en el empleo de unos conjuntos de nodos que sirven para reducir, drásticamente, el número de evaluaciones heurísticas, lo cual podría mejorar, espectacularmente, el tiempo de respuesta.

En este artículo se presentará el algoritmo BHFFA*, una implementación admisible del algoritmo BHFFA que mejora, sustancialmente, el tiempo de otros algoritmos de búsqueda unidireccionales y bidireccionales. En la sección siguiente se muestra el algoritmo BHFFA, y en la sección 3, el algoritmo BHFFA* que será probado, en la sección 4, sobre diferentes dominios. El artículo acaba, en la sección 5, con la presentación de las conclusiones.

2 El algoritmo BHFFA

El algoritmo BHFFA, propuesto por Champeaux y colaboradores [Champeaux, 1977] es un algoritmo de el mejor primero que realiza dos búsquedas simultáneamente. Pero, en vez de dirigir cada búsqueda hacia el otro extremo, como hace el BHPA, cada nodo considera su distancia hasta los nodos terminales de la búsqueda realizada en sentido contrario.

Para describir más detalladamente el algoritmo BHFFA, cuyo pseudocódigo se muestra en la figura 1, es necesario introducir primero los siguientes términos:

(1) Poner s en $\tilde{S}$ y t en $\tilde{T}$. Hacer $S = T = \emptyset$.
(2) Si $\tilde{S} \cup \tilde{T} = \emptyset$, parar con fallo. En otro caso, decidir ir hacia delante (paso 3), o hacia detrás (paso 7).
(3) Seleccionar el nodo $n \in \tilde{S}$, para el que $f_s(n) = min_{x \in \tilde{S}}\{f_s(x)\}$.
(4) Si $n \in \tilde{T}$ parar con la solución formada por la concatenación de los caminos P^*_{s-n} y P^*_{n-t}.
(5) Si SCS$(n) = \emptyset$, ir al paso (2).
(6) $\forall x \in$ SCS(n), añadirlo a $\tilde{S}$, con un puntero hacia n. Ir al paso (2).
(7) Repetir los pasos (3) a (6) cambiando (S, $\tilde{S}$, $\tilde{T}$, s, t, x, y) por (T, $\tilde{T}$, $\tilde{S}$, t, s, y, x).

Figura 1: Pseudocódigo del algoritmo BHFFA

S	Conjunto de nodos alcanzables desde s que ya han sido expandidos.
T	Conjunto de nodos alcanzables desde t que ya han sido expandidos.
$\tilde{S}$	Conjunto de nodos alcanzables desde s que aún no han sido expandidos. Son los nodos terminales de la búsqueda hacia delante. Se denomina, genéricamente, *frontera de la búsqueda hacia delante*.
$\tilde{T}$	Conjunto de nodos alcanzables desde t que aún no han sido expandidos. Son los nodos terminales de la búsqueda hacia detrás. Se denomina, genéricamente, *frontera de la búsqueda hacia detrás*.
x	Un nodo cualquiera perteneciente a $\tilde{S}$.
y	Un nodo cualquiera perteneciente a $\tilde{T}$.
$g_s(n)$	Coste del camino que une el nodo inicial s con n, desarrollado en la búsqueda hacia delante.
$g_t(n)$	Coste del camino que une el nodo final t con n, desarrollado en la búsqueda hacia detrás.
$f_s(n)$	$= g_s(n) + min_{y \in \tilde{T}}\{h(n,y) + g_t(y)\}$
$f_t(n)$	$= g_t(n) + min_{x \in \tilde{S}}\{h(n,x) + g_s(x)\}$
SCS(n)	Conjunto de sucesores del nodo n.

Así, el algoritmo BHFFA expande aquel par de nodos x e y que minimicen la cantidad:

$$f(x,y) = g_s(x) + h(x,y) + g_t(y) \tag{1}$$

o, si se quiere, que expande aquel nodo $x \in \tilde{S}$, tal que:

$$f_s(x) = min_{n \in \tilde{S}}\{f_s(n)\} \tag{2}$$

y análogamente para el nodo $y \in \tilde{T}$. El algoritmo BHFFA finaliza tan pronto como hay una coincidencia entre ambas fronteras, con la solución que resulta de concatenar el camino desde el nodo inicial hasta el nodo común y desde éste hasta el nodo final.

Como se ve, en el paso (2) es posible elegir el sentido de la búsqueda. Esta capacidad del algoritmo BHFFA permite la aplicación del *principio de cardinalidad* de Pohl [Pohl, 1969] que establece que puede reducirse el tiempo de búsqueda, eligiendo siempre un nodo de la frontera que tenga la lista abierta más pequeña.

El primer inconveniente de este algoritmo es que es necesario considerar, en cada paso, todos los pares de nodos x e y de ambas fronteras para obtener, en el paso (3), el menor $f_s(n)$ —o $f_t(n)$, según corresponda. Por lo tanto, si el número necesario de comparaciones es muy alto, el rendimiento del algoritmo se degradará sustancialmente.

El segundo inconveniente —mucho más grave— es que este algoritmo no es admisible, es decir, aunque la función heurística empleada, $h(x,y)$, sea admisible, el algoritmo puede terminar con una solución no óptima. La figura 2(a) muestra un grafo en el que el algoritmo BHFFA elegiría el camino $\langle$s, t$\rangle$, con un coste de 3 unidades, en vez del camino $\langle$s, X, t$\rangle$, con un coste de 2

Figura 2: El algoritmo BHFFA no es admisible

unidades. La figura 2(b) representa un grafo donde, tal vez, no se encuentre la solución óptima si la coincidencia en el nodo Y se detecta antes que en el nodo X. La primera figura está tomada del artículo que, en 1983, escribiera Champeaux [Champeaux, 1983] presentando las correcciones necesarias para que el algoritmo sea admisible. A esta segunda versión la denominó BHFFA2. Desafortunadamente, este algoritmo es tan complicado que parece casi irrealizable. Básicamente consiste en dos bucles:ENCUENTRA-UN-CAMINO y ENCUENTRA-EL-MEJOR-CAMINO. El primer bucle hace avanzar las dos búsquedas hasta que ambas coinciden en algún nodo usando las reglas que se muestran en las ecuaciones 1 y 2. Una vez que ambas fronteras han colisionado, se pasa al segundo bucle, del que no se puede volver al primero, y donde se busca el camino óptimo. Además, esta segunda versión del algoritmo presenta diferentes procedimientos para la expansión de nodos: una para el primer bucle y otra, diferente y mucho más complicada, para el segundo. Por añadidura, aún no se han publicado resultados de la aplicación del algoritmo BHFFA2 y sus prestaciones nunca se han contrastado con otros algoritmos.

Por todos estos motivos, el algoritmo BHFFA, y su variante BHFFA2, han caído completamente en desuso.

3 El algoritmo BHFFA*

Como se mencionó anteriormente, cualquiera de los algoritmo BHFFA o BHFFA2, permiten la aplicación de un número cualquiera de pasos hacia delante o hacia detrás. Sin embargo, si se impone como estrategia de búsqueda alternar consecutivamente entre ambos sentidos de búsqueda, el algoritmo BHFFA puede formularse utilizando una heurística consistente, de manera admisible, y en términos muy sencillos. A cambio, sólo se sacrifica la realización de políticas como la del principio de cardinalidad.

El algoritmo BHFFA* [Linares, 1997] expande, alternativamente, aquel nodo x, para el que $f_s(x)$ sea menor y el nodo y, para el que $f_t(y)$ sea menor. El algoritmo se detiene cuando los nodos x e y han coincidido, es decir, son el mismo y, además, ocurre que $f_s(x) = g_s(x) + g_t(y)$, si la coincidencia se detectó en la búsqueda hacia delante, o $f_t(y) = g_t(y) + g_s(x)$, si la coincidencia ocurrió en la búsqueda hacia detrás.

Además, el algoritmo BHFFA* *no realiza todos los productos cruzados*, es decir, no considera en cada iteración todos los pares de nodos x e y, sino que, en su lugar, sólo considera las distancias de los descendientes del nodo expandido en una frontera con todos los nodos de la frontera contraria.

El pseudocódigo del algoritmo se muestra en la figura 3.

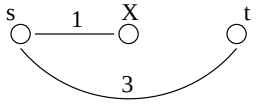
Nótese que:

- En el paso (1), el cálculo de $f_s(s)$ y $f_t(t)$, no es estrictamente necesario, pero se añade para no perder la generalidad en caso de que los nodos s y t sean el mismo.
- La inicialización de los conjuntos S y T en el paso (1) y la incorporación del nodo n a S, en el paso (5), se hace para mantener la consistencia de las estructuras definidas, pero en la práctica no es necesario implementar los conjuntos S y T y, en lo sucesivo, no se volverán a considerar.

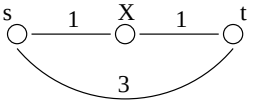
(1) Poner s en $\tilde{S}$ y t en $\tilde{T}$. Hacer $S = T = \emptyset$. Calcular $f_s(s)$ y $f_t(t)$.
(2) Si $\tilde{S} \cup \tilde{T} = \emptyset$, parar con fallo.
(3) Si $\tilde{S} = \emptyset$, ir al paso (9).
(4) Coger el primer nodo $n \in \tilde{S}$ (para el que se cumplirá que $f_s(n) = min_{x \in \tilde{S}}\{f_s(x)\}$).
(5) Si $n \notin \tilde{T}$, poner n en S y eliminarlo de $\tilde{S}$.
(6) Si $n \in \tilde{T}$
(6.1) Si $f_s(n) = g_s(n) + g_t(n)$, parar con la solución formada por la concatenación de los caminos P^*_{s-n} y P^*_{n-t}.
(6.2) Si $f_s(n) \neq g_s(n) + g_t(n)$, hacer $f_s(n) = g_s(n) + min\{g_t(n)\}$.
(7) Si SCS$(n) = \emptyset$, ir al paso (9).
(8) $\forall x \in$ SCS(n), calcular $f_s(x) = g_s(x) + min_{y \in \tilde{T}}\{h(x,y) + g_t(y)\}$ e insertarlo, en orden creciente de $f_s(x)$, en $\tilde{S}$.
(9) Cambiar el sentido de la búsqueda. Esto es, repetir los pasos (2) a (8) cambiando (S, $\tilde{S}$, $\tilde{T}$, s, t, x, y) por (T, $\tilde{T}$, $\tilde{S}$, t, s, y, x).

Figura 3: Pseudocódigo del algoritmo BHFFA*

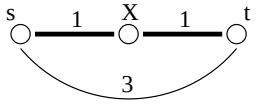
(a) Generación de los nodos inicial y final



(b) Expansión del nodo s



(c) Expansión del nodo t



(d) Finalización

Figura 4: Algoritmo de búsqueda BHFFA*

- Por último, si un nodo n pertenece a $\tilde{T}$, no se elimina de $\tilde{S}$ (paso (5)). Esto se hace así para permitir que, en el futuro, se considere el camino cerrado que ahora se ha descubierto, y cuyo valor se establece en (6.2), pero cuya admisibilidad no puede garantizarse en este momento. Nótese que el nuevo valor de $f_s(n)$ —para este caso— no incluye el valor de $h(\cdot)$ puesto que, como ha habido coincidencia, $h(n,n) = 0$.

Como ejemplo de su ejecución, considérese el caso que se muestra en la figura 2(a) que, como se mencionó anteriormente, no está correctamente resuelto por el algoritmo BHFFA. Los pasos para su resolución se muestran en la figura 4. Para ello, se ha considerado una heurística consistente "adhoc" cualquiera, para la que:

$$h(s,t) = h(t,s) = 3 \leq h^*(s,t) = h^*(t,s) = 3$$
$$h(X,s) = h(X,t) = 1 \leq h^*(X,s) = h^*(X,t) = 1$$
$$h(s,X) = h(t,X) = 1 \leq h^*(s,X) = h^*(t,X) = 1$$

además, los valores de $g(\cdot)$ se muestran en los arcos del grafo.

Inicialmente, se crean los nodos inicial y final, como se muestra en la figura 4(a), y se añaden a los conjuntos $\tilde{S}$ y $\tilde{T}$, respectivamente, quedando $\tilde{S} = \{s\}$ y $\tilde{T} = \{t\}$. Además:

$$\begin{aligned} f_s(s) &= g_s(s) + min_{y \in \tilde{T}}\{h(s,y) + g_t(y)\} = \\ &= 0 + min\{h(s,t) + g_t(t)\} = \\ &= 0 + min\{3 + 0\} = 3 \end{aligned}$$

y:

$$\begin{aligned} f_t(t) &= g_t(t) + min_{x\in\tilde{S}}\{h(t,x) + g_s(x)\} = \\ &= 0 + min\{h(t,s) + g_s(s)\} = \\ &= 0 + min\{3+0\} = 3 \end{aligned}$$

A continuación, se ejecuta el paso (4), que lleva a la situación mostrada en la figura 4(b) en la que $n = s$. Los valores de la función de evaluación para todos los sucesores del nodo seleccionado (paso (8)) son:

$$\begin{aligned} f_s(X) &= g_s(X) + min_{y\in\tilde{T}}\{h(X,y) + g_t(y)\} = \\ &= 1 + min\{h(X,t) + g_t(t)\} = \\ &= 1 + min\{1+0\} = 2 \\ f_s(t) &= g_s(t) + min_{y\in\tilde{T}}\{h(t,y) + g_t(y)\} = \\ &= 3 + min\{h(t,t) + g_t(t)\} = \\ &= 3 + min\{0+0\} = 3 \end{aligned}$$

que se añaden, en orden creciente de $f_s(\cdot)$ a $\tilde{S}$ que ahora contiene los nodos $\{X, t\}$ en este orden, puesto que $f_s(X) \leq f_s(t)$. Nótese que el nodo s ya no pertenece a $\tilde{S}$, porque fue eliminado del conjunto en el paso (5).

La figura 4(c), muestra la ejecución de una nueva iteración en sentido contrario, es decir, buscando a partir del nodo final, t[1]. En el paso (4), se toma el nodo $n = t$. En este caso, $n \in \tilde{S}$, pero $f_t(t) = 3 \neq min_{x\in\tilde{S}}\{f_s(x)\} = f_s(X) = 2$, por lo que *el algoritmo no finaliza* (el algoritmo BHFFA hubiera finalizado, exactamente, en este punto, con una solución incorrecta). A continuación, se procede con normalidad y, para todos los sucesores de t —X y s— se calcula el valor de $f_t(\cdot)$:

$$\begin{aligned} f_t(X) &= g_t(X) + min_{x\in\tilde{S}}\{h(X,x) + g_s(x)\} = \\ &= 1 + min\{h(X,X) + g_s(X), h(X,t) + g_s(t)\} = \\ &= 1 + min\{0+1, 1+3\} = 2 \\ f_t(s) &= g_t(s) + min_{x\in\tilde{S}}\{h(s,x) + g_s(x)\} = \\ &= 3 + min\{h(s,X) + g_s(X), h(s,t) + g_s(t)\} = \\ &= 3 + min\{1+1, 3+3\} = 5 \end{aligned}$$

de modo que, habiéndose eliminado el nodo t de $\tilde{T}$, en el paso (5), queda ahora en el paso (8), $\tilde{T} = \{X, s\}$, insertados en este orden puesto que $f_t(X) < f_t(s)$.

Nuevamente se vuelve a cambiar el sentido de la búsqueda, de modo que ahora se considera el conjunto $\tilde{S}$. En el paso (4), se toma el nodo X, y en el paso (6), se observa que este nodo pertenece también a $\tilde{T}$, dándose además la circunstancia de que $f_s(X) = min_{y\in\tilde{T}}\{f_t(y)\} = f_t(X) = 2$ por lo que la solución es el camino $\langle$s, X, t$\rangle$ con un coste de 2 unidades, tal y como se muestra en la figura 4(d).

A continuación se muestran varios lemas y teoremas que prueban la admisibilidad del algoritmo BHFFA* y, además, explican su comportamiento. En todos los teoremas y lemas que siguen, así como en sus demostraciones, se supone siempre la admisibilidad de la función heurística $h(\cdot)$ (es decir, $h(x,y) \leq h^*(x,y) \quad \forall x, y$). Además se supone, sin perdida de generalidad, que sólo hay una solución (y que necesariamente existe alguna). Si no fuera así, en todos los teoremas, lemas y demostraciones que siguen, sólo debe considerarse un nodo de entre todos aquellos cuyo valor $f_s(\cdot)$ o $f_t(\cdot)$, según corresponda, es igual al de la solución óptima.

Teorema 1 BHFFA* *es un algoritmo completo, es decir, necesariamente termina con una solución.*

Demostración. Si el algoritmo no finalizara, sería porque, o bien está recorriendo un camino infinito, o bien termina con un error, o bien no hay coincidencia entre ambas fronteras.

[1]Se advierte que ahora se han alterado los símbolos en el sentido indicado en el paso (9)

El primer caso no es posible, puesto que el coste de un camino infinito es infinito. Debido a que todos los arcos tienen un coste estrictamente positivo y finito, necesariamente siempre habrá otros nodos con un valor finito que serán expandidos antes que él.

El segundo caso tampoco es posible. Para ello, debiera ocurrir que $\tilde{S} \cup \tilde{T} = \emptyset$ (paso (2)). Como ocurre que desde un principio los conjuntos $\tilde{S}$ y $\tilde{T}$ tienen, al menos, los nodos inicial y final (paso (1)), la unión será vacía sólo si después de haber expandido todos los nodos de ambas fronteras, no hubo descendientes (paso (8)), lo que es imposible puesto que, como por hipótesis existe una solución, cualquier nodo de dicho camino debe tener, al menos, un descendiente que le conduzca hasta el extremo final y un antecesor hasta el nodo inicial.

Por otra parte, si existe un camino entre los nodos s y t, necesariamente ambas fronteras deberán coincidir en algún momento, puesto que para la expansión de un nodo se considera siempre su proximidad a todos los de la frontera contraria, tomando aquél que minimice el esfuerzo total. Si dos trayectorias, muy cercanas, se cruzaran sin tocarse, dejarán entonces de expandirse cuando la suma de sus costes más la estimación heurística de su proximidad (que aumentará en cada iteración, puesto que ya se han cruzado) sea superior que la de algún otro par de nodos (que necesariamente deben existir puesto que se supone que existe una solución), los cuales comenzarán entonces a expandirse.

Si cuando la coincidencia se detecta, se cumple que $f_s(n) = g_s(n)+g_t(n)$ necesariamente debe concluirse que se ha encontrado un camino entre los nodos inicial y final que, como cumple las condiciones de los pasos (6) y (6.1), terminará con la ejecución del algoritmo. Por el contrario, si en el momento de la coincidencia, sucede que $f_s(n) \neq g_s(n) + g_t(n)$, el valor de $f_s(n)$ se actualizará al coste del camino desde el nodo inicial hasta el nodo n y el menor coste desde el nodo final hasta el mismo nodo. Como dicho nodo no se elimina de la frontera, tal y como se mencionó anteriormente, el nuevo valor de $f_s(n)$ será considerado en las nuevas elecciones de nodos, de modo que si vuelve a ser seleccionado cumplirá la condición del paso (6.1) y terminará con la ejecución del algoritmo.

Por lo tanto, el algoritmo BHFFA* terminará, necesariamente, con alguna solución. □

Lema 1 *En cualquier momento, antes de que el algoritmo* BHFFA* *termine, existen nodos* $n_s \in \tilde{S}$ *y* $m_t \in \tilde{T}$*, pertenecientes al camino óptimo,* $P^*(s,t)$*, para los cuales* $f_s(n_s) \leq h^*(s,t)$ *y* $f_t(m_t) \leq h^*(s,t)$*.*

El mismo lema fue probado para los algoritmos BHFFA y BHFFA2 de modo que la demostración para el algoritmo BHFFA* es como la de aquellos.

Demostración. Si se considera el camino $\langle s, n_1, n_2, \ldots, n_s \rangle$ perteneciente al camino óptimo, construido en la búsqueda hacia delante, y el camino $\langle t, m_1, m_2, \ldots, m_t \rangle$ perteneciente también al camino óptimo, construido en la búsqueda hacia detrás, tal que $n_s \in \tilde{S}$ y $m_t \in \tilde{T}$, se tiene que:

$$
\begin{array}{ll}
f_s(n_s) = g_s(n_s) + min_{y \in \tilde{T}}\{h(n_s, y) + g_t(y)\} = & \text{(por definición de } f_s\text{)} \\
= g_s(n_s) + h(n_s, y) + g_t(y) \leq & \text{(para algún } y \in \tilde{T}\text{)} \\
\leq g_s(n_s) + h(n_s, m_t) + g_t(m_t) \leq & \text{(porque } m_t \in P^*_{s-t}\text{)} \\
\leq g_s(n_s) + h^*(n_s, m_t) + g_t(m_t) = & \text{(por la admisibilidad de h)} \\
= h^*(s,t) &
\end{array}
$$

Obsérvese que los nodos n_s y m_t necesariamente deben existir y pertenecer a $\tilde{S}$ y $\tilde{T}$ respectivamente, o de lo contrario, el algoritmo BHFFA* no podría terminar con una solución (puesto que los nodos a expandir se escogen de dichos conjuntos), lo que contradice el teorema 1.

La demostración para $f_t(m_t)$ es análoga. □

Teorema 2 *El algoritmo* BHFFA* *no expande, en ningún momento, un nodo* $n \in \tilde{S}$ *o* $m \in \tilde{T}$*, para el cual* $f_s(n) > h^*(s,t)$ *o* $f_t(m) > h^*(s,t)$*.*

Demostración. Como el Lema 1 garantiza que siempre habrá, al menos, un nodo perteneciente a cada frontera con un valor menor o igual que $h^*(s,t)$, y, además, como el algoritmo BHFFA* toma siempre el nodo con menor valor, éste nunca podrá ser mayor que $h^*(s,t)$. □

Teorema 3 *El algoritmo* BHFFA* *expande todos los nodos* $n \in \tilde{S}$*, para los cuales* $f_s(n) \leq h^*(s,t)$*.*

Demostración. La demostración se hace por inducción sobre la cardinalidad del conjunto:

$$\mathcal{U}_s = \{x \in \tilde{S} | f_s(x) \leq h^*(s,t)\}$$

con los nodos ordenados ascendentemente por su valor $f_s(x)$. Como el Lema 1 garantiza que siempre hay al menos un nodo con un valor inferior o igual que el del coste de la solución, el conjunto $\mathcal{U}_s$ no puede ser nunca vacío.

Así, el caso base es $|\mathcal{U}_s| = 1$. En este caso, el nodo n tomado para su expansión, pertenece a $\mathcal{U}_s$, ya que este conjunto contiene al nodo con menor $f_s(\cdot)$. Como ningún otro nodo pertenece a dicho conjunto, en este caso se concluye que todos los nodos con un coste inferior o igual al del camino óptimo, han sido expandidos.

Ahora, asumiendo que se cumple el teorema para un tamaño $|\mathcal{U}_s| = k - 1$, considérese un nodo n, de entre los que forman el nuevo conjunto $\mathcal{U}_s$, cuya cardinalidad sea k. Si, en el paso (6) del pseudocódigo del algoritmo BHFFA*, se encuentra que dicho nodo pertenece a la frontera contraria, $\tilde{T}$, y $f_s(n) = g_s(n) + g_t(n)$, necesariamente se ha encontrado la solución, puesto que por hipótesis (de pertenencia a $\mathcal{U}_s$), $f_s(n) \leq h^*(s,t)$ y, de hecho, se ha formado un camino que une s y t, por lo que $f_s(n) = h^*(s,t)$. Así, todos los nodos que, en orden ascendente, van después de n, tendrán un coste superior a $h^*(s,t)$. Porque cuando se expandió el nodo n, se expandieron también todos los que van antes que él (porque BHFFA* toma los nodos para su expansión, en el orden de su valor) y los que van después, tienen un valor superior a $h^*(s,t)$, queda demostrado que se han expandido todos los nodos cuyo coste es menor o igual que el del camino óptimo. □

Teorema 4 *El algoritmo* BHFFA* *expande todos los nodos* $n \in \tilde{T}$*, para los cuales* $f_t(n) \leq h^*(s,t)$*.*

Demostración. Análoga a la del teorema anterior. □

Teorema 5 BHFFA* *es un algoritmo admisible.*

La demostración del siguiente teorema es análoga a la prueba de admisibilidad del A* elaborada por Pearl [Pearl, 1984], lo cual no debe sorprender, puesto que el BHFFA* puede considerarse como una versión bidireccional del algoritmo A*.

Demostración. Supóngase que el algoritmo BHFFA* termina con la expansión de un nodo $x \in \tilde{S}$ (la pertenencia a $\tilde{S}$ se hace sin pérdida de generalidad, y el caso de la pertenencia a $\tilde{T}$ es absolutamente análogo) que cumple las condiciones de terminación formuladas en el paso (6) del pseudocódigo del algoritmo BHFFA*, para el que $f_s(x) > h^*(s,t)$ (es decir, el algoritmo finaliza con una solución no óptima). Puesto que este nodo fue elegido para ser expandido, forzosamente cumple que:

$$f_s(x) \leq f_s(n) \quad \forall n \in \tilde{S}$$

y, por ello, $f_s(n) > h^*(s,t) \quad \forall n \in \tilde{S}$, lo cual contradice el Lema 1, que asegura que, al menos, hay un nodo cuyo valor $f_s(n)$ es inferior o igual al coste del camino óptimo. Por ello, cuando el algoritmo finaliza, lo hace con un coste igual al del camino óptimo. □

4 Resultados

Para estudiar la eficiencia del algoritmo BHFFA*, se han tomado dos problemas diferentes: el grafo del Metro de Madrid y el juego del N-'Puzle', y se han aplicado los algoritmos A*[Hart, 1968], IDA* [Korf, 1985], RBFS [Korf, 1993], BIDA* [Manzini, 1995] y BHFFA*[Linares, 1997].

Además, para garantizar la igualdad de condiciones en las comparaciones de los resultados proporcionados por la ejecución de los diferentes algoritmos, se ha intentado reutilizar tanto código como sea posible en su programación, de modo que ninguna implementación pueda

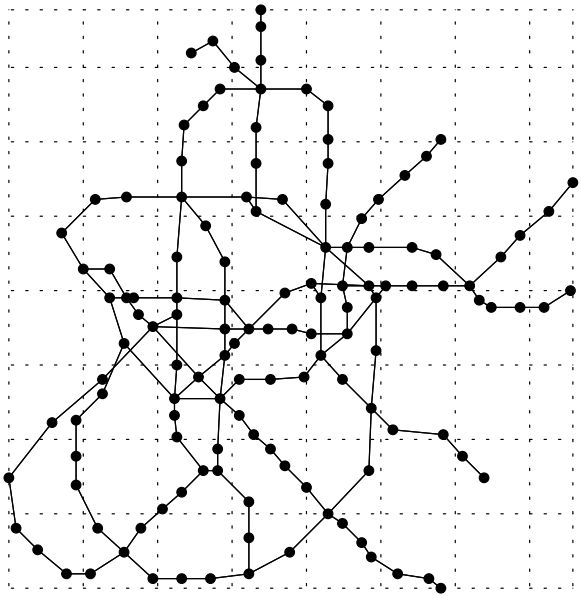


Figura 5: Red del Metro de Madrid

beneficiarse de mejoras o refinamientos relativos, únicamente, a características intrínsecas del dominio sobre el que se aplican.

Los problemas elegidos se corresponden con la distinción entre dominios discretos y continuos. Los dominios discretos se caracterizan porque el coste de sus arcos es siempre una cantidad natural o discreta, mientras que los dominios continuos tienen costes reales.

4.1 El grafo del Metro de Madrid

El grafo del Metro de Madrid, mostrado en la figura 5, tiene 133 nodos y un factor de ramificación puro de 2,3. Además, este grafo tiene la riqueza suficiente como para estudiar problemas con soluciones que tienen hasta 25 nodos.

En total, se eligieron 100 pares de instancias diferentes, cuyo histograma de distancias heurísticas se muestra en la figura 6. La distancia heurística media es de 75,75 unidades, y el rango que se cubre va desde las 9,21 unidades (instancia número 70) hasta las 161,307 unidades (instancia número 11). Se observa, además, que la moda está cerca de las 50 unidades. Esto es así porque, como se puede apreciar en el grafo del Metro de Madrid, mostrado en la figura 5, la mayor densidad de nodos ocurre en el centro del grafo por lo que, tomando aleatoriamente diferentes pares de nodos, la probabilidad de que ambos pertenezcan al centro es mayor que la de cualquier otro caso y, por ello, la distancia euclídea será menor.

A diferencia del juego del N-'Puzle', este tipo de dominios no han sido demasiado tratados, salvo, tal vez, para los problemas de enrutamiento de vehículos y otros que, con frecuencia, se redactan como problemas de optimización de una función de evaluación sujeta a varias restricciones [Insua, 1988]. El grafo del Metro de Madrid pertenece a la clase de dominios *continuos* puesto que el coste de cruzar un arco es igual a la distancia de los nodos que separa, y ésta es una cantidad real o continua. La función heurística empleada para resolver un caso es igual a la distancia euclídea entre un par de nodos, $h(u,v) = d_E(u(x_1,y_1), v(x_2,y_2)) = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$, y esta cantidad también es continua.

La tabla 1 muestra las mediciones de todos los algoritmos implementados relativas a memoria y tiempo consumido sobre 100 pares de nodos tomados aleatoriamente. El algoritmo BHFFA* resolvió el peor caso (para una diferencia heurística de 145 unidades) en menos de un cuarto de segundo (0,21 segundos), con un consumo de memoria igual a 55836 bytes. Como puede verse, este algoritmo fue muy superior, en término medio,

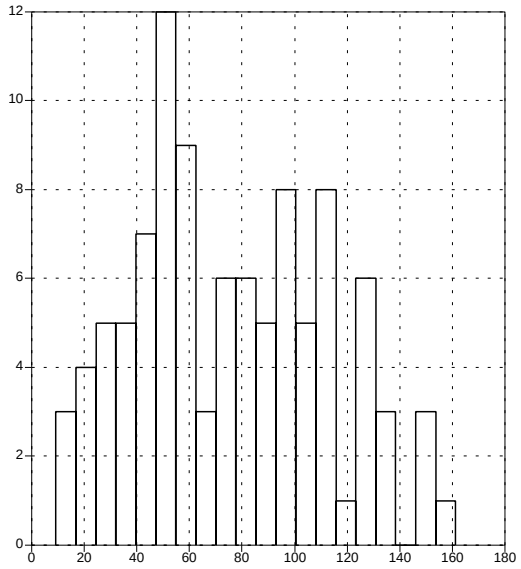


Figura 6: Histograma de distancias heurísticas del Metro

Algoritmo	Tiempo (segundos)		Memoria (bytes)	
	Media	Varianza	Media	Varianza
A*	0,319	1,151	186241,33	414622505641,684
IDA*	42,6	26891,673	9256188,02	1269955206415126,0
$BIDA^*_{20}$	5,774	497,0	2197707,491	72471355762353,375
$BIDA^*_{40}$	3,449	180,109	1249483,425	23546669418620,738
$BIDA^*_{60}$	0,802	6,796	304841,622	902432537124,645
$BIDA^*_{80}$	0,407	0,139	248670,649	76120276172,104
$BIDA^*_{100}$	1,38	4,421	1165601,574	4128961547876,363
BHFFA*	0,05	0,002	14862,341	190733517,975
RBFS	16,661	3820,403	4394485,424	257045216111203,437

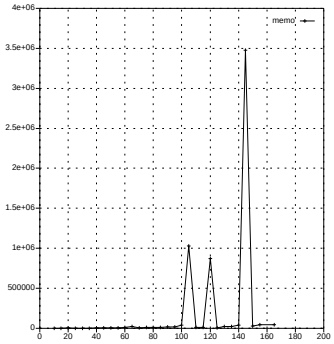
Tabla 1: Observaciones del grafo del Metro de Madrid

al resto.

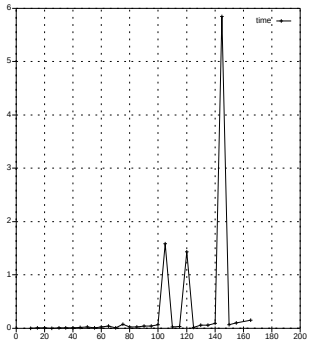
El algoritmo A* dedicó, para la resolución del peor caso 3477516 bytes durante 5,85 segundos. Por lo tanto, la reducción lograda con el BHFFA* ha sido, en este caso, superior al 95%, tanto en términos de memoria como de tiempo. La figura 7 muestra el consumo de tiempo y memoria del A* para todos los casos, en funcin de la distancia heurstica entre los nodos inicial y final. La figura 8 muestra los resultados con el algoritmo BHFFA*.

El IDA* fue el más lento de todos los algoritmos: consumió, para el peor de todos los casos, 893,62 segundos y dedicó algo más de 185 Mbytes de memoria. La reducción de los recursos computaciones con el BHFFA* es, en este caso, superior al 99%. Esta diferencia no debe sorprender, puesto que ya se ha demostrado [Patrick, 1989] que la complejidad algorítmica del IDA* es de $O(n^2)$ en los dominios continuos, donde n es el número de nodos generados por el algoritmo de búsqueda mejor primero.

Asimismo, el algoritmo BHFFA* resulto ser muy superior al algoritmo RBFS, que tardó, en el peor de los casos, 335,82 segundos y utilizó 82,5 Mbytes de memoria. Como en el caso anterior, la reducción del algoritmo BHFFA* fue superior también al 99%.

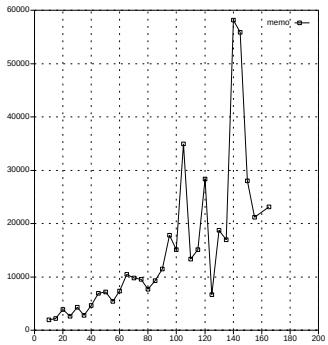


(a) Memoria consumida (en bytes)

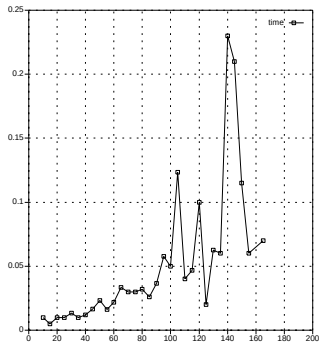


(b) Tiempo consumido (en segundos)

Figura 7: Mediciones del algoritmo A*



(a) Memoria consumida (en bytes)



(b) Tiempo consumido (en segundos)

Figura 8: Mediciones del algoritmo BHFFA*

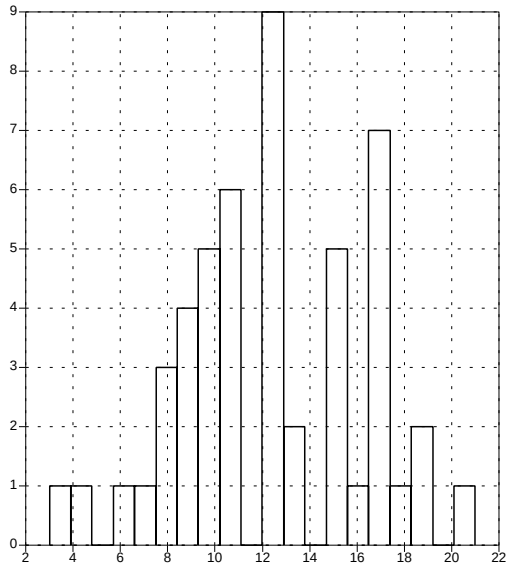


Figura 9: Histograma de distancias heurísticas del 15-Puzle

Además, se probaron cinco valores de perímetro diferentes con el algoritmo BIDA*: 20, 40, 60, 80 y 100. De ellos, el mejor fue BIDA^*_{80}, que resolvió el peor caso en tan solo 0,38 segundos empleando 151228 bytes. Sin embargo, este algoritmo fue, en término medio, 8 veces más lento y consumió 16 veces más memoria que el BHFFA*.

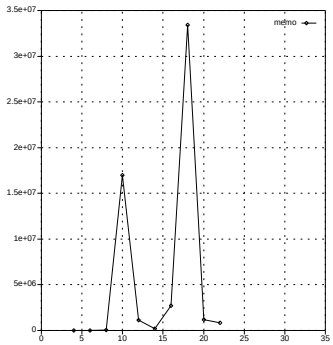
4.2 El juego del N-'Puzle'

El juego del rompecabezas es, sin ningún lugar a dudas, el dominio más usado en los experimentos sobre búsqueda —del que existen innumerables referencias. El juego del N-'Puzle' consiste en un tablero de $N \times N$ posiciones, sobre las que se colocan $N^2 - 1$ fichas numeradas, de modo que se deja una sola posición vacía. En cada tablero, sólo se pueden desplazar aquellas posiciones que son vertical u horizontalmente adyacentes a la posición vacía. El objeto del juego consiste en disponer las fichas en un orden concreto. Para la realización de los experimentos se tomará un tablero de 4×4 fichas. Al juego que resulta se le denomina 15-'Puzle'.

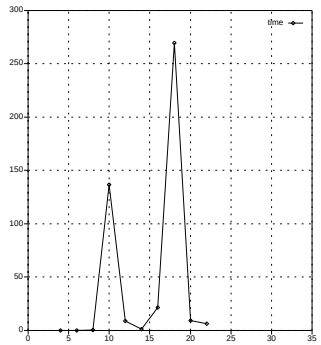
Aunque pueda parecerlo, no se trata de un problema trivial: el espacio de estados del 15-'Puzle' es de $16!/2 = 10.461.394.944.000 \approx 10^{13}$ posiciones [Culberson, 1994] y el factor de ramificación puro es 3. Además, se ha demostrado que el N-'Puzle' es un problema NP-duro [Ratner, 1986].

Este problema pertenece a la clase de los dominios *discretos*, porque el coste de cada arco es siempre el mismo e igual a la unidad, y esta es una cantidad natural o discreta. Por otra parte, la función heurística empleada es la *distancia Manhattan*, que es igual a la suma, para todas las fichas del tablero excluyendo la posición vacía, del número de movimientos que como mínimo deben hacerse para colocar cada ficha mal situada en el orden que le corresponde en el nodo meta.

Para la realización de las pruebas, se generaron aleatoriamente 50 instancias. El histograma de la distribución de los valores heurísticos entre los nodos inicial y final, se muestra en la figura 9. La media de la distancia heurística es 12,34 y, como se ve en el histograma, su moda es también 12. Aunque estos problemas son mucho más sencillos que las 100 instancias resueltas por Korf, servirán para poner de manifiesto la calidad



(a) Memoria consumida (en bytes)



(b) Tiempo consumido (en segundos)

Figura 10: Mediciones del algoritmo BHFFA*

de cada algoritmo y las diferencias que hay entre ellos.

Como es bien sabido, el principal inconveniente del algoritmo A* es, como el de todos los algoritmos de el mejor primero (salvo el RBFS), su alto consumo de memoria. Debido a ello, el A* no resolvió ni siquiera la mitad de los problemas propuestos, por lo que no es posible ofrecer resultados de su ejecución. Sin embargo, el BHFFA* sí los resolvió todos, aunque a un coste muy alto, puesto que en este dominio el número de nodos generados es mucho mayor y, por ello, la cantidad de evaluaciones heurísticas aumenta en exceso. La tabla 2 (de la que se ha excluído el RBFS, puesto que Korf ya ha estudiado exhaustivamente este dominio con dicho algoritmo [Korf, 1993]) muestra el consumo medio de memoria y tiempo para la resolución de 50 casos escogidos aleatoriamente. La figura 10 muestra grficamente los consumos de tiempo y memoria del algoritmo BHFFA*.

Sin embargo, en este caso, el BHFFA* no fue, ni mucho menos, el mejor algoritmo. El IDA* consumió, aproximadamente, 20 veces menos recursos computaciones que él. Además, mientras que el tiempo medio para la resolución de las 50 instancias fue de 52,23 segundos, todos los perímetros probados con el algoritmo BIDA* (2, 4 y 6) ofrecieron mejores resultados.

Algoritmo	Tiempo (segundos)		Memoria (bytes)	
	Media	Varianza	Media	Varianza
IDA*	2,138	4,169	394175,408	120551885583,673
$BIDA^*_2$	1,576	4,318	262298,493	122790455798,054
$BIDA^*_4$	2,226	6,042	353350,016	131618356242,437
$BIDA^*_6$	4,899	12,585	905039,19	347330248927,162
BHFFA*	52,232	3654,896	6501953,12	56070462336522,46

Tabla 2: Observaciones del 15-'Puzle'

5 Conclusiones

En este artículo se ha presentado el algoritmo BHFFA*, una implementación bidireccional del algoritmo A*, que consiste en el conjunto de modificaciones necesarias del algoritmo BHFFA para garantizar su admisibilidad.

Este algoritmo ha sido probado en dos dominios diferentes y, en cualquier caso, el A* resultó ser mucho menos eficiente que el BHFFA*, al contrario que el BS* (otra implementación bidireccional del A*), que es menos eficiente que su versión unidireccional.

Se ha observado que el BHFFA* funciona muy bien en un dominio continuo, donde la variabilidad de los costes perjudica tanto a otros algoritmos como el IDA* o el RBFS. Sin embargo, en el dominio discreto propuesto en este artículo, los resultados fueron inferiores a los del resto de los algoritmos. Ello se debe, fundamentalmente, a la cantidad de evaluaciones heurísticas que, en este escenario, debe realizar el BHFFA*. Aunque el nuevo algoritmo no compara todos los pares de nodos x e y pertenecientes a cada frontera (como el BHFFA), realiza una cantidad de evaluaciones heurísticas muy alta. Si se considera la magnitud del espacio de estados del 15-'Puzle', y el hecho de que el factor de ramificación es mayor que en el caso del grafo del Metro de Madrid, resultará inmediato comprender que este algoritmo no es una buena elección si el tiempo de evaluación heurística es, como ocurre en el N-'Puzle', alto (el tiempo de evaluación de la distancia Manhattan fue, en término medio, 25 veces más lenta que la distancia heurística). En otro caso, el BHFFA* puede resultar en reducciones de tiempo y memoria superiores al 99%.

Referencias

[Champeaux, 1983] Champeaux, D. D. (1983). Bidirectional heuristic search again. *Journal of the Association for Computing Machinery*, volumen 30, número 1, páginas 22–32.

[Champeaux, 1977] Champeaux, D. D.; Sint, L. (1977). An improved bidirectional heuristic search algorithm. *Journal of the Association for Computing Machinery*, volumen 24, número 2, páginas 177–191.

[Culberson, 1994] Culberson, J. C.; Schaeffer, J. (1994). Efficiently searching the 15-puzzle. Technical Report TR 94-08, University of Alberta.

[Dillenburg, 1994] Dillenburg, J. F.; Nelson, P. C. (1994). Perimeter search. *Artificial Intelligence*, volumen 65, páginas 165–178.

[Hart, 1968] Hart, P. E., Nilsson, N. J.,; Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernet.*, volumen 4, número 2, páginas 100–107.

[Insua, 1988] Insua, S. R. (1988). *Investigación Operativa*. Centro de Estudios Ramón Areces, Madrid.

[Korf, 1985] Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, volumen 27, páginas 97–109.

[Korf, 1993] Korf, R. E. (1993). Linear-space best-first search. *Artificial Intelligence*, volumen 62, páginas 41–78.

[Kwa, 1989] Kwa, J. B. H. (1989). BS*: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence*, volumen 38, páginas 95–109.

[Linares, 1997] Linares, C. (1997). Algoritmos de búsqueda de un agente en dominios discretos y continuos. Tesis de Master, Facultad de Informática. Universidad Politécnica de Madrid.

[Manzini, 1995] Manzini, G. (1995). BIDA*: an improved perimeter search algorithm. *Artificial Intelligence*, volumen 75, páginas 347–360.

[Nilsson, 1987] Nilsson, N. J. (1987). *Principios de Inteligencia Artificial*. Ediciones Díaz de Santos, Madrid.

[Patrick, 1989] Patrick, B. G., Almulla, M.,; Newborn, M. M. (1989). An upper bound on the complexity of iterative-deepening-A*. En *Proceedings Symposium on Artificial Intelligence and Mathematics*.

[Pearl, 1984] Pearl, J. (1984). *Heuristics*. Addison-Wesley, Reading MA.

[Pohl, 1969] Pohl, I. (1969). *Bi-directional and Heuristic Search in Path Problems*. Stanford University, Stanford, CA. SLAC Report 104.

[Pohl, 1987] Pohl, I. (1987). *Encyclopedia of Artificial Intelligence*, volumen 2, capítulo Search, Bidirectional, página 1000. John Wiley & Sons.

[Politowski, 1984] Politowski, G.; Pohl, I. (1984). D-node retargeting in bidirectional heuristic search. En *Proceedings of the Fourth AAAI Conference*, páginas 274–277.

[Ratner, 1986] Ratner, D.; Warmuth, M. (1986). Finding a shortest solution for the n×n extension of the 15-puzzle is intractable. En *Proceedings AAAI-86*, páginas 168–172.