Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Linares López, Carlos
Busqueda bidireccional en dominios discretos y continuos
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 4, núm. 10, verano, 2000, p. 0
Asociación Española para la Inteligencia Artificial
Valencia, España

Búsqueda bidireccional en dominios discretos y continuos

Carlos Linares López*
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
28660 Boadilla del Monte - Madrid
clinares@delicias.dia.fi.upm.es

Resumen

En este trabajo se muestra que es posible superar los principales inconvenientes de la búsqueda bidireccional y desarrollar nuevos algoritmos admisibles, con una heurística consistente, y en términos muy sencillos. La conclusión más relevante de la investigación que se presenta en este trabajo es que, contra toda intuición, los algoritmos de búsqueda bidireccionales presentados sirven para encontrar soluciones óptimas con un consumo de tiempo y memoria inferior al de sus versiones unidireccionales. Por ello, se anima a la comunidad científica a que recupere el interés por estos algoritmos.

1 Introducción

A pesar de que inicialmente hubo un gran interés en los algoritmos de búsqueda bidireccionales (véase la introducción de [15]), 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 [10, 15].

En este trabajo se muestra, sin embargo, que es posible superar los principales inconvenientes de la búsqueda bidireccional y desarrollar nuevos algoritmos admisibles, con una heurística consistente, y en términos muy sencillos. Además, a diferencia de otras implementaciones bidireccionales, en este trabajo se muestra cómo es posible mejorar incluso en un 99% los resultados obtenidos con algoritmos unidireccionales. Para constatarlo, se han estudiado dos dominios radicalmente diferentes: el grafo del Metro de Madrid y el juego del N-'Puzle'. En cualquier caso, los algoritmos tratados se restringen al generalísimo caso de costes finitos.

Las aportaciones más importantes consisten en la presentación de cuatro nuevos algoritmos de búsqueda bidireccional que, en el caso del algoritmo BHFFA*, está basada en otro algoritmo bidireccional (el BHFFA[2]) y, en los otros, en el algoritmo de búsqueda unidireccional RBFS [13]. Para mejorar la comprensión de este trabajo, ambos algoritmos serán someramente presentados antes de describir las innovaciones que se proponen.

Las secciones 2–5 presentan los nuevos algoritmos de búsqueda, junto con una cantidad suficiente de justificaciones teóricas sobre su admisibilidad. La sección 6 muestra los resultados obtenidos que sirven, en la sección 7, para la obtención de interesantes conclusiones. Por último, el trabajo concluye en la sección 8 con la presentación de algunas líneas futuras de in-

*Dirección actual: INtelligent ADvisors (INAD). Centro Empresarial Euronova. Ronda de Poniente, 2. 28760 Tres Cantos - Madrid. clinares@inad.es

vestigación.

Por último, este trabajo ha sido premiado *ex-equo* —junto a D. José Salvador Sánchez Garreta— por la Asociación Española Para la Inteligencia Artificial (AEPIA), en la categoría de mejor trabajo predoctoral en Torremolinos, en Noviembre de 1 997, durante el transcurso de la Séptima Conferencia de la Asociación Española Para la Inteligencia Artificial.

2 BHFFA*

En 1977, Dennis de Champeaux y colaboradores [2] propusieron el algoritmo BHFFA (acrónimo de *Bidirectional Heuristic Front-to-Front Algorithm*) que consiste en desarrollar, simultáneamente, dos búsquedas: la primera, desde el nodo inicial; la segunda, desde el nodo final. Sin embargo, en vez de dirigirse al extremo opuesto, lo hacen a aquél nodo contrario para el que se minimice el coste del camino entre los nodos inicial y final. Es decir, se trata simplemente de un algoritmo de el mejor primero que expande aquel par de nodos x e y, x hacia delante e y hacia detrás, que minimizan la cantidad:

$$f(x,y) = g(x) + h(x,y) + g(y) \qquad (1)$$

La expansión se realiza hasta que ambas fronteras coinciden, en cuyo caso, la solución consiste en la concatenación de los dos caminos que han llevado hasta el nodo intermedio desde los nodos inicial y final, respectivamente.

Uno de los inconvenientes más importantes de este algoritmo es el de la gran cantidad de evaluaciones heurísticas que deben hacerse, puesto que *cada nodo se compara con todos los nodos terminales desarrollados en la búsqueda contraria*. Dicha cantidad crece exponencialmente en cada iteración y su consumo podría llegar a ser prohibitivo.

Sin embargo, el inconveniente más grave que tiene, es que no encuentra soluciones óptimas, es decir, no es admisible. Por ello, Dennis de Champeaux presentó, en 1983, un nuevo algoritmo llamado BHFFA2 [1] con las correcciones necesarias para que el algoritmo sea admisible. 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 la regla que se muestra en la ecuación 1. 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.

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 se puede formular utilizando una heurística consistente, de manera admisible, y en términos muy sencillos.

Para describir detalladamente el algoritmo BHFFA* [14, 15], es necesario introducir primero los siguientes términos:

- s Nodo inicial
- t Nodo final
- 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 $\tilde{T}$.

(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) En otro caso, 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 1: Pseudocódigo del algoritmo BHFFA*

$g_s(n)$	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.
$h(n,m)$	Estimación heurística del coste del camino que une los nodos n y m.
$P^*(n,m)$	Camino óptimo desde el nodo n hasta m.
$h^*(n,m)$	Coste del camino $P^*(n,m)$.
$f_s(n)$	$= g_s(n) + min_{y \in \tilde{T}}\{h(n,y) + g_t(y)\}$
$f_t(n)$	$= g_t(n) + min_{x \in \tilde{S}}\{h(n,x) + g_s(x)\}$
SCS(n)	Conjunto de sucesores del nodo n.

El algoritmo BHFFA* 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 1.

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.
- 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.

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 hay una y sólo una solución. 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. El algoritmo BHFFA* expande siempre aquel nodo perteneciente a cada frontera cuyo valor $f_s(\cdot)$ o $f_t(\cdot)$, según corresponda, sea menor, y considera la pertenencia de sus sucesores a la frontera contraria. Si se considera el instante, inmediatamente anterior al momento en que se encuentra la solución, después de la expansión del nodo n, se cumple que: primero, el sucesor del nodo n, n_i, pertenece a la frontera contraria y, segundo:

$$f_s(n_i) = g_s(n_i) + g_t(n_i) \quad (\text{porque } h(n_i, n_i) = 0)$$

Por lo que se cumple la condición de terminación del paso (6.1) y el algoritmo finaliza con una solución.

Por otra parte, si el algoritmo no finalizara, sería porque, o bien está recorriendo un camino infinito, o bien no hay coincidencia entre ambas fronteras.

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, necesariamente siempre habrá otros nodos con un valor finito que serán expandidos antes que él.

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.

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* $n \in \tilde{S}$ *y* $m \in \tilde{T}$, *para los cuales* $f_s(n) \leq h^*(s,t)$ *y* $f_t(m) \leq h^*(s,t)$.

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:

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

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 inferior a $h^*(s,t)$, 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 siguiente nodo, n, tomado para su expansión, pertenecerá también 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 [18], 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 $f_s(x) > h^*(s,t)$. 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. □

3 RBFPS*

Este algoritmo es el primer intento de implementar el algoritmo RBFS [12, 13] de manera bidireccional.

Para explicar el algoritmo RBFS se introducirá, primero, el algoritmo SRBFS (*Simple Recursive Best-First Search*), ambos inventados por Korf.

El algoritmo SRBFS considera para cada nodo un umbral local (en contraposición con los algoritmos IDA* [11] y BIDA* [16], que utilizan umbrales globales) que, para el nodo inicial, es siempre $+\infty$. A continuación genera sus sucesores y los ordena por su valor $f(n) = g(n) + h(n)$ (como todos los algoritmos de el mejor primero). A partir de entonces, expande siempre el mejor nodo que haya en la lista abierta y explora todo el subárbol de búsqueda que haya bajo él, hasta que todos los nodos terminales de dicho subárbol excedan el umbral $\eta_n = f(n)$. El mínimo de los excesos cometidos será el nuevo valor del nodo n y, si fuera necesario, su posición se modificará en la lista abierta.

El umbral local del mejor sucesor de un nodo, será el mínimo entre el umbral local de su padre, y el de la puntuación de su mejor hermano puesto que: primero, no deben explorarse nodos cuya puntuación, $f(n)$, exceda el umbral local del padre; segundo, porque el mejor descendiente lo será mientras su puntuación no exceda la de su mejor hermano, en cuyo caso, la búsqueda

(1) Realizar una búsqueda a profundidad d, a partir del nodo final, t. Almacenar el perímetro, P_d.
(2) Aplicar el algoritmo RBFS al nodo inicial, s, utilizando la función heurística $h_d(n)$ sobre el conjunto de nodos $\mathcal{P}_d(n, \eta)$, donde η es el umbral local del nodo n.

Figura 2: Pseudocódigo del algoritmo RBFPS*

debería continuar por él. Así se opera, sucesivamente, hasta que, por fin, se alcance el nodo final con un coste no mayor que el del umbral establecido (es decir, los algoritmos SRBFS y RBFS son eminentemente unidireccionales).

Aunque este algoritmo encuentra soluciones óptimas con una ocupación lineal de memoria, aunque la función de coste sea no monótona, es ineficiente puesto que, cada vez que se "re-expande" un nodo, ocurrirán las mismas actualizaciones de los valores umbrales para todos sus descendientes y el flujo de recorrido del árbol de búsqueda será siempre el mismo. Es decir, a cada "re-expansión" le sigue la repetición de un trabajo que ya se hizo con anterioridad.

Por lo tanto, el algoritmo RBFS consiste en las modificaciones necesarias para evitar la repetición de dicho trabajo. La diferencia fundamental consiste en que cada nodo tendrá ahora, además del valor $f(n)$, denominado *valor estático* otro, $F(n)$, conocido como *valor almacenado*. El primer valor es, simplemente, la cantidad $g(n) + h(n)$ y $F(n)$ será el mínimo exceso cometido por los nodos terminales del subárbol de búsqueda que hay por debajo del nodo n. Por lo tanto, si el valor almacenado de un nodo es mayor que su valor estático, será porque ya ha sido expandido anteriormente, y su valor $F(n)$ es un límite inferior de la puntuación del subárbol de búsqueda que hay bajo él, de modo que a sus descendientes se les asigna un valor igual al máximo entre el valor almacenado de su padre y sus propios valores estáticos:

$$F(n_i) = max\{F(n), f(n_i)\} \qquad (2)$$

Si, por el contrario, el valor estático de un nodo es igual a su valor almacenado, será porque aún no ha sido expandido, de modo que sus hijos tendrán un valor igual a su valor estático.

$$F(n_i) = f(n_i)$$

Por otra parte, el valor del nuevo umbral se determina de la misma manera que en el algoritmo SRBFS, salvo que ahora debe considerarse el valor almacenado, $F(n_j)$, del mejor nodo hermano n_j, en vez de su valor estático $f(n_j)$:

$$\eta_{n_i} = min\{\eta_n, F(n_j)\} \quad |n_i, n_j \in \mathrm{SCS}(n)$$

y el proceso, como en aquel caso, se repite hasta que por fin se alcance el estado final con un coste no mayor que el del umbral establecido.

El algoritmo RBFPS* [14] (acrónimo de *Recursive Best-First Perimeter Search*) consiste en utilizar el algoritmo RBFS en la búsqueda hecha a partir del nodo inicial en los algoritmos de perímetro [4], con la función heurística $h_d(n)$, definida de la manera:

$$h_d(n) = \min_{m \in P_d} \{h(n, m) + h^*(m, t)\}$$

donde P_d es el conjunto de los nodos alcanzables desde el extremo final, t, con un coste menor o igual que d y que, además, tienen descendientes que se encuentran a una distancia de t estrictamente mayor que d.

Para ello, se ha empleado la misma técnica que la utilizada por el algoritmo BIDA* [16] para la reducción del número de evaluaciones heurísticas. Esto es, el mantenimiento de un conjunto $\mathcal{P}_d(n, \eta_n)$, para cada nodo n, donde η_n es el umbral local de dicho nodo, o el coste máximo que puede consumirse en la construcción de cualquier trayectoria que pase por él.

El pseudocódigo del algoritmo RBFPS* se muestra en la figura 2, en el que se han obviado los detalles de la implementación del algoritmo RBFS. Dado que se trata de un algoritmo de perímetro, como cualquier otro, es necesario proporcionar el parámetro d, o amplitud del perímetro.

La admisibilidad de este algoritmo resulta de la admisibilidad del algoritmo que se aplique en la búsqueda hacia delante [4], y está demostrada por Korf [13]. Aunque para el algoritmo RBFS no es necesario, además se da la circunstancia de que la función heurística $h_d(n)$ es admisible [16].

4 BRBFS*

Si es posible implementar una versión de perímetro utilizando el algoritmo RBFS [13] en la búsqueda a partir del nodo inicial, tal y como sucede en el algoritmo RBFPS*, y teniendo en cuenta que el perímetro consiste en todos los nodos que, *en cualquier dirección*, están a una distancia menor o igual que una cantidad fija y predeterminada d, también debe ser posible desarrollar un algoritmo que considere su distancia hasta un conjunto de nodos pero, que en vez de ser fijos como es el caso del perímetro, sea otro conjunto que también se aproxima, y que es variable.

Esta es la idea principal del algoritmo BRBFS* [14] (acrónimo de *Bidirectional Recursive Best-First Search*). Este algoritmo aplica, alternativamente, una búsqueda desde el nodo inicial y otra desde el nodo final con el algoritmo RBFS, salvo que, en vez de considerar una heurística convencional que mide la distancia a un nodo, se aplica una heurística que estime la distancia hasta el conjunto de nodos terminales de la búsqueda realizada en sentido contrario.

Por lo tanto, el algoritmo BRBFS* considera dos fronteras diferentes: F_s y F_t. La primera consiste en el conjunto de nodos terminales de la búsqueda que se hace a partir del nodo inicial, s, y la segunda es el conjunto de nodos terminales de la búsqueda que se hace a partir del nodo final, t. Formalmente:

$$F_s = \{n | n \text{ es terminal y es alcanzable desde } s\}$$
$$F_t = \{n | n \text{ es terminal y es alcanzable desde } t\}$$

Además, para cada nodo n almacenado en cualquiera de las dos fronteras, se guarda su camino hasta el nodo inicial o final —según corresponda— y el coste de dicho camino, $h^*(n,s)$ o $h^*(n,t)$. Nótese que esta información siempre estará disponible, puesto que a estos nodos se llega aplicando el algoritmo de búsqueda RBFS.

Así, las funciones heurísticas aplicadas son:

$$h_s(n) = min_{m \in F_t}\{h(n,m) + h^*(m,t)\}$$
$$h_t(n) = min_{m \in F_s}\{h(n,m) + h^*(m,s)\}$$

donde $h_s(n)$ se utiliza en la búsqueda hacia delante, y $h_t(n)$ en la búsqueda hacia detrás. Por ello, los valores *estáticos* serán:

$$f_s(n) = g(n) + h_s(n)$$
$$f_t(n) = g(n) + h_t(n)$$

y los valores *almacenados* se obtienen de la misma manera que en el algoritmo RBFS [13]. Esto es, si el valor almacenado de un nodo es mayor que su valor estático, sus sucesores tendrán:

$$F_s(n_i) = max\{F_s(n), f_s(n_i)\}$$

y en otro caso:

$$F_s(n_i) = f_s(n_i)$$

y análogamente para $F_t(n)$.

Como el algoritmo RBFS considera, para cada nodo, un umbral que no será mayor para ninguno de sus descendientes (puesto que cada sucesor tiene un umbral local igual al mínimo entre el umbral de su padre y el valor de su mejor hermano), es posible definir los conjuntos:

$$\mathcal{P}_s(n,\eta_n) = \{m \in F_t | g(n) + h(n,m) + h^*(m,t) \leq \eta_n\}$$
$$\mathcal{P}_t(n,\eta_n) = \{m \in F_s | g(n) + h(n,m) + h^*(m,s) \leq \eta_n\}$$

donde η_n es el umbral local del nodo n. Obsérvese que, como en el caso del BIDA* [16], $f_s(n)$ será mayor que η_n sí y sólo sí el conjunto $\mathcal{P}_s(n,\eta_n)$ es vacío. Análogamente, $f_t(n)$ será mayor que η_n sí y sólo sí el conjunto $\mathcal{P}_t(n,\eta_n)$ es vacío. Por lo tanto, la función heurística $h_s(n)$ sólo debe aplicarse sobre los nodos del conjunto $\mathcal{P}_s(n,\eta_n)$, y la función

heurística $h_t(n)$ sobre los nodos del conjunto $\mathcal{P}_t(n, \eta_n)$, puesto que para el resto, $f_s(n)$ y $f_t(n)$ serán, respectivamente, mayores que el umbral.

Además, para cualquier camino $\langle n_1,\ n_2,\ \ldots,\ n_k\rangle$ se cumple que:

$$\mathcal{P}_s(n_i, \eta_{n_i}) \subseteq \mathcal{P}_s(n_{i-1}, \eta_{n_{i-1}})$$
$$\mathcal{P}_t(n_i, \eta_{n_i}) \subseteq \mathcal{P}_t(n_{i-1}, \eta_{n_{i-1}})$$

puesto que sólo puede ocurrir que, o bien $\eta_{n_i} = \eta_{n_{i-1}}$, o $\eta_{n_i} < \eta_{n_{i-1}}$. En el primer caso, se cumple que:

$$g(n_{i-1}) + h(n_{i-1}, m) + h^*(m, t) =$$
$$= g(n_i) - c(n_i, n_{i-1}) + h(n_{i-1}, m) + h^*(m, t) \leq$$
$$\leq g(n_i) + h(n_i, m) + h^*(m, t)$$

sí y sólo sí, $h(n, m)$ es una heurística monótona. En el segundo caso, la inecuación anterior se sigue cumpliendo, y además, como $\eta_{n_i} < \eta_{n_{i-1}}$ puede haber nodos pertenecientes al conjunto $\mathcal{P}_s(n_{i-1}, \eta_{n_{i-1}})$ para los cuales $f(\cdot)$ es mayor que η_{n_i}.

Las expresiones anteriores se cumplen, igualmente, para el conjunto $\mathcal{P}_t(n, \eta_n)$.

Por lo tanto, el conjunto $\mathcal{P}_s(n, \eta_n)$ *decrece en cada iteración* y, por lo tanto, el número de evaluaciones heurísticas también. Sin embargo, se advierte que ha sido necesario renunciar a la no monotonía de la función heurística $h(\cdot)$, lo que en el caso del RBFS no era necesario.

Así, el algoritmo BRBFS* funciona de la siguiente manera: inicialmente, invoca el procedimiento BUSCAR —que, como se verá, consiste en una adaptación del algoritmo RBFS— sobre el nodo inicial s, con la frontera $F_t = \{t\}$, y un valor almacenado igual a $h_s(s)$, que será igual a $h(s, t)$ (puesto que el coste del nodo inicial es nulo). El umbral η_s toma el mismo valor. Como el umbral del nodo inicial es finito (a diferencia del algoritmo RBFS), y necesariamente menor que la distancia real entre los nodos inicial y final (puesto que la heurística $h(\cdot)$ debe ser monótona, y por ello admisible), el procedimiento BUSCAR finalizará en algún momento devolviendo un valor mayor que el umbral inicial del nodo s. A continuación, el algoritmo BRBFS* vuelve a invocar su procedimiento auxiliar, pero esta vez sobre el nodo final t, y un valor almacenado y umbral iguales al valor devuelto en la iteración anterior. La frontera que se considerará en este caso, será F_s, el conjunto de nodos terminales de la búsqueda anterior. Nuevamente, el procedimiento finalizará en algún momento, devolviendo un valor mayor que el valor almacenado que recibió, y otra vez se vuelve a invocar el mismo procedimiento sobre el nodo inicial s, la frontera F_t generada en la iteración anterior, y un valor almacenado y un umbral iguales al valor devuelto por la última ejecución del procedimiento BUSCAR. Así sucesivamente, hasta que por fin ambas fronteras coinciden en un nodo n, en cuyo caso la solución consiste en la concatenación de los caminos que llevan desde el nodo s hasta n y desde n hasta t.

El pseudocódigo del procedimiento BUSCAR se muestra en la figura 3, y el de BRBFS* en la figura 4. Puesto que este algoritmo se expresa muy bien de manera recursiva, se ha elegido una representación funcional.

El procedimiento BUSCAR devuelve, o bien un camino solución (paso (2)), o bien un nuevo umbral (paso (15)). Por ello, el algoritmo BRBFS* realiza llamadas a este procedimiento hasta que, por fin, el valor devuelto es la solución buscada. Mientras tanto, el valor devuelto (y almacenado en Π) sirve como umbral para la siguiente iteración. Puesto que este umbral necesariamente debe crecer, cada búsqueda se acerca un poco más a la frontera contraria, hasta que por fin se tocan (paso (2) de BUSCAR). Además, cada llamada a BUSCAR se hace sobre el conjunto $\mathcal{F}$ de nodos terminales de la búsqueda contraria, cuya computación se prepara, siempre, en la iteración anterior con el conjunto $\mathcal{Q}$. Si al principio de una nueva búsqueda se inicializa este conjunto al vacío, es suficiente comprobar para cada nuevo nodo n_i, si su conjunto $\mathcal{P}$ (que, en la búsqueda hacia delante, será $\mathcal{P}_s$ y, en la búsqueda hacia detrás, será $\mathcal{P}_t$) es vacío, o no (paso (6) de BUSCAR). Si es así, el valor de este nodo ha excedido su umbral, η_{n_i}, y se añade al conjunto de nodos terminales (paso (7)). Al final de la iteración actual, este conjunto será el conjunto $\mathcal{F}$ de la siguiente iteración.

Como se puede ver, BUSCAR es un algoritmo RBFS, salvo que:

```
      BUSCAR(n, F(n), η_n, F)
(1)   IF f(n) > η_n THEN RETURN f(n)
(2)   IF n ∈ F THEN EXIT BUSCAR
(3)   IF SCS(n)=∅ THEN RETURN +∞
(4)   FOR cada hijo n_i ∈ SCS(n)
(5)         Calcular P(n_i, η_n) = {m ∈ F | g(n_i) + h(n_i, m) + h*(m) ≤ η_n}
(6)         IF P(n_i, η_n) = ∅ THEN
(7)               Añadir n_i a Q.
(8)         IF f(n) < F(n) THEN F(n_i) = max{F(n), f(n_i)}
(9)         ELSE F(n_i) = f(n_i)
(10)  Ordenar n_i en orden creciente de F(n_i)
(11)  IF |SCS(n)|=1 THEN F(n_2) = +∞
(12)  WHILE (F(n_1) ≤ η_n AND F(n_1) < +∞)
(13)        F(n_1)=BUSCAR(n_1, F(n_1), min{η_n, F(n_2)}, P(n_1, η_n))
(14)        Insertar n_1 en orden creciente según F(n_1)
(15)  RETURN F(n_1)
```

Figura 3: Pseudocódigo del procedimiento BUSCAR

```
      BRBFS*(s, t)
(1)   F = {t}, Q = ∅, Π = h_s(s)
(2)   WHILE (Π no sea un camino solución)
(3)         Π=BUSCAR(s, Π, Π, F)
(4)         Intercambiar s y t
(5)         F = Q, Q = ∅
(6)   RETURN Π
```

Figura 4: Pseudocódigo del procedimiento BRBFS*

- El nodo raíz no se invoca con un límite igual a $+\infty$, sino que se utiliza una cantidad finita. Si dicho límite es inferior al coste real entre los nodos s y t, el algoritmo finalizará en algún momento devolviendo un valor mayor que el umbral inicial.
- Tiene los mismos parámetros que RBFS y, además, el conjunto de nodos $\mathcal{F}$, conjunto de nodos terminales de la búsqueda realizada en sentido contrario en la última iteración. Inicialmente será, únicamente, el nodo final, t. Nótese que, como se alterna sucesivamente el sentido de la búsqueda, en una iteración, $\mathcal{F}$ será el conjunto F_s y, en la siguiente, F_t.

Por lo demás, BUSCAR se comporta como RBFS: mantiene valores almacenados por cada nodo, cuya inicialización y actualización se hace de la misma manera que en el RBFS, así como el umbral local de cada nodo.

Por otra parte, la admisibilidad de este algoritmo se fundamenta en la admisibilidad del algoritmo RBFS, demostrada por Korf [13]. Así, para probar la admisibilidad del algoritmo BRBFS*, se utilizarán los mismos lemas que los empleados para el RBFS. A continuación, se introducen las definiciones y lemas suficientes para ello.

Definición 1 *Se denomina subárbol de búsqueda, T(n), a aquél cuya raíz es el nodo n, y tal que si el nodo m pertenece a T(n), también pertenecerán todos sus hermanos.*

Definición 2 *Se denomina subárbol de búsqueda limitado a η_n, $T(n, \eta_n)$, al subárbol de T(n), para cuyos nodos interiores $f(\cdot) \leq \eta_n$, y para cuyos nodos terminales $f(\cdot) > \eta_n$.*

Definición 3 *Se denomina $MF(n, \eta_n)$, al*

mínimo valor estático, $f(\cdot)$, de los nodos terminales del subárbol de búsqueda $T(n, \eta_n)$.

Obsérvese que $\mathrm{MF}(n, \eta_n)$ será siempre mayor que η_n, puesto que todos los nodos terminales de $\mathrm{T}(n, \eta_n)$ tienen, por definición, un valor mayor que η_n.

Lema 2 *Todas las llamadas al procedimiento BUSCAR son de la manera BUSCAR(n, $F(n)$, η_n, $\mathcal{F}$), donde $\eta_n \leq F(n)$.*

Demostración. Inicialmente, el algoritmo BRBFS* invoca el procedimiento BUSCAR con $\eta_n = F(n) = h_s(s)$ (línea 1). Además, las sucesivas llamadas se hacen con $\eta_n = F(n) = \Pi$. Por otra parte, por inspección del código del procedimiento BUSCAR, se ve que las llamadas recursivas son de la manera BUSCAR(n_1, $F(n_1)$, $min\{\eta_n, F(n_2)\}$, $\mathcal{P}(n_1, \eta_n)$), y como forzosamente $min\{\eta_n, F(n_2)\} \leq \eta_n$, se concluye que para todas las llamadas recursivas se cumple que $\eta_n \leq F(n)$. □

Lema 3 *Si η_n es un límite finito, y $T(n, \eta_n)$ no contiene ningún nodo final, entonces BUSCAR$(n, F(n), \eta_n, \mathcal{F})$ recorre $T(n, \eta_n)$ y devuelve MF(n, η_n)*

Demostración. Puesto que el algoritmo BUSCAR es exactamente igual que el algoritmo RBFS, salvo para el cálculo de la función heurística, para la que se considera la frontera de la búsqueda contraria, el resultado de Korf [13] se aplica a este caso igualmente. □

Definición 4 *Se denomina OD(n) al mínimo valor estático, $f(\cdot)$, de todos los nodos, en la lista abierta, que son descendientes del nodo n.*

Definición 5 *Se denomina ON(n) al mínimo valor estático, $f(\cdot)$, de todos los nodos, en la lista abierta, que no son descendientes del nodo n.*

Lema 4 *Para todas las invocaciones BUSCAR$(n, F(n), \eta_n, \mathcal{F})$, se cumple que $F(n) \leq OD(n)$ y que $\eta_n \leq ON(n)$.*

Demostración. Como antes, el resultado de Korf [13] puede aplicarse igualmente aquí. □

Lema 5 *Cuando BUSCAR expande un nodo, su valor estático, $f(\cdot)$, es menor o igual que el valor estático de todos los nodos de la lista abierta.*

Demostración. Como antes, puesto que BUSCAR expande los nodos de la misma manera que RBFS, aquí se puede aplicar igualmente, el resultado de Korf [13]. □

Teorema 6 *BRBFS(s, t) realizará varias búsquedas mejor primero, unas veces a partir de s y otras, a partir de t, hasta que se detiene devolviendo la solución óptima.*

Demostración. El lema 5 prueba que BUSCAR realiza una búsqueda mejor primero, y puesto que el BRBFS* consiste, simplemente, en sucesivas llamadas a dicho procedimiento, se concluye que BRBFS* realiza varias búsquedas mejor primero.

Por otra parte, el lema 3, asegura que cada llamada a BUSCAR devuelve el mínimo valor estático de todos los nodos terminales del nuevo árbol de búsqueda (es decir, el mínimo de los excesos cometidos). Como en cada nueva llamada, el límite η_n será igual al exceso de la iteración anterior, cada invocación devuelve valores estrictamente mayores, haciendo que cada vez se examinen árboles de búsqueda a mayor profundidad. Y así sucesivamente, de modo que del bucle `while` de BRBFS* sólo se puede salir si se encuentra una solución.

Además, la solución encontrada debe ser, necesariamente, la solución óptima. Según el lema 5, cada nodo m que se considera para ser expandido tiene el menor valor estático, $f(m)$, de todos los nodos de la lista abierta. Además, dicho valor será menor que el umbral η_m o, de lo contrario, este nodo sería el nodo terminal de un árbol de búsqueda mayor. Por ello, si el nodo m coincide con la frontera $\mathcal{F}$ se puede asegurar que el coste desde el nodo raíz hasta m, más el coste hasta el otro extremo (que está almacenado en la frontera), es el menor valor de todos los nodos de la lista abierta y, por ello, es la solución óptima. □

5 IBRBFS*

Probablemente, uno de los mayores inconvenientes del algoritmo BRBFS*, consiste en la gran cantidad de re-expansiones que realiza, puesto que cada dos iteraciones vuelve a comenzar la búsqueda desde el mismo extremo (en unos casos s y, en otros, t).

En realidad, puesto que es necesario mantener en memoria la frontera de una búsqueda, debiera ser posible modificar el algoritmo anterior para que mantenga, simultáneamente, las fronteras de la búsqueda hacia delante y hacia detrás.

Éste es, precisamente, el modo de operar del algoritmo IBRBFS* [14] (acrónimo de *Improved Bidirectional Recursive Best-First Search*). Este algoritmo pretende ahorrar el mayor número de expansiones —y con ello, de evaluaciones heurísticas—, manipulando las fronteras obtenidas en la iteración anterior, de modo que pueda comenzar la búsqueda a partir de los nodos que contienen, en vez de hacerlo desde el extremo inicial o final, otra vez.

Como quiera que cada vez que se añade un nodo a la frontera es porque su valor estático excedió su umbral η_n (o sea, no es expandido), en las siguientes iteraciones el valor almacenado, $F(n)$, será igual a su valor estático, $f(n)$, por lo que, para la programación del IBRBFS*, sólo es necesario mantener en cada frontera el valor estático de cada nodo, $f(n)$, y, si se quiere, su trayectoria hasta su nodo raíz, a fin de construir luego la solución.

Además, también con el ánimo de mejorar tanto como sea posible este nuevo algoritmo bidireccional, se usó el principio de cardinalidad de Pohl [20], que recomienda continuar la búsqueda por el subárbol que tenga el menor factor de ramificación, en vez de hacerlo alternativamente. Obsérvese que, en el algoritmo IBRBFS*, esto se traduce como continuar la búsqueda por la frontera más pequeña, lo que es extremadamente fácil de programar y no tendrá ningún consumo adicional. Este principio se fundamenta en la observación, de que el menor factor de ramificación es siempre menos costoso e, inevitablemente, la expansión de su frontera servirá para acercar ambos extremos.

Por último, su admisibilidad se deriva de la del algoritmo BRBFS*, puesto que las acciones realizadas a partir de los nodos de cada frontera son, en cualquier caso, exactamente las mismas.

6 Resultados

La programación del entorno de pruebas se ha realizado siguiendo el estándar Common Lisp, que consiste en un conjunto de recomendaciones realizadas por el subcomité X3J13 de ANSI, y recopiladas por Steele [9]. Para la programación se ha elegido el compilador/intérprete CLisp de Haible, Stoll y Daniels, disponible como software GNU. El seguimiento del estándar, garantiza que el mismo código puede compilarse en otros muchos compiladores e intérpretes (como Allegro Common Lisp o Liquid Common Lisp) y que, por lo tanto, se puede ejecutar sobre diferentes sistemas (Linux, Sun, AIX, ...).

Para estudiar la eficiencia de los algoritmos presentados, 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* [5], IDA* [11], RBFS [13], BIDA* [16], BHFFA* [14, 15], RBFPS* [14], BRBFS* [14] e IBRBFS* [14].

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 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.

6.1 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 que, en el centro del grafo —el núcleo de la ciudad de Madrid—, llega a

ser, aproximadamente, 3,3. Además, este grafo tiene la riqueza suficiente como para estudiar problemas con soluciones que tienen hasta 25 nodos.

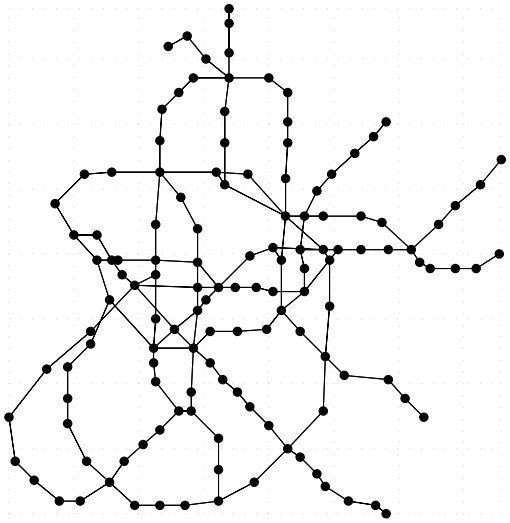


Figura 5: Grafo del Metro de Madrid

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 [6]. 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.

El histograma de la figura 6 muestra las frecuencias relativas de las distancias euclídeas entre 100 pares de nodos tomados aleatoriamente.

La tabla 1 muestra los recursos computacionales consumidos, en término medio, por cada algoritmo implementado, junto con las varianzas de cada observación. Como se ve, la eficiencia del BHFFA* es muy superior a la del resto de los algoritmos.

La figura 7 muestra los resultados obtenidos con

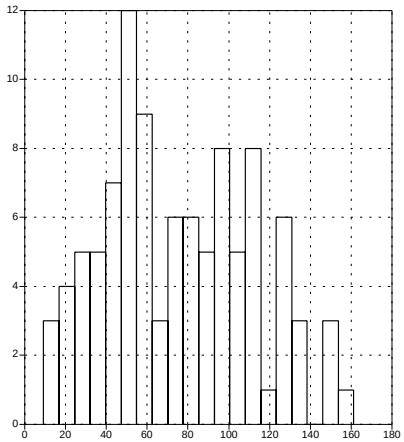


Figura 6: Histograma de distancias heurísticas del Metro

los mejores algoritmos aplicados relativos a memoria y tiempo consumido. Como puede verse en la figura, el peor caso (para una diferencia heurística de 145 unidades) es resuelto por el BHFFA* en menos de un cuarto de segundo (0,21 segundos), con un consumo de memoria igual a 55836 bytes. Este resultado es muy superior al obtenido con cualquier otro algoritmo. Además, el segundo mejor algoritmo, el RBFPS^*_{80}, es otro de los propuestos en este trabajo. Este algoritmo tarda, en término medio cerca de 8 veces más que el BHFFA* y consume, aproximadamente, 16 veces más memoria.

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.

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. Asimismo, la diferencia en término medio es superior al 99% tanto en términos de memoria consumida como de tiempo dedicado. Esta diferencia no debe sorprender, puesto que ya se ha demostrado [17] 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.

Además, el algoritmo BHFFA* resultó ser muy superior al algoritmo RBFS, que tardó, en el

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
RBFS	16,661	3820,403	4394485,424	257045216111203,437
$BHFFA^*$	0,05	0,002	14862,341	190733517,975
$RBFPS^*_{20}$	4,002	238,668	1669270,423	40495725511715,125
$RBFPS^*_{40}$	2,179	70,129	910922,406	12007314581659,287
$RBFPS^*_{60}$	0,538	2,634	227595,248	448830399128,208
$RBFPS^*_{80}$	0,391	0,145	237345,839	77466212681,254
$RBFPS^*_{100}$	1,387	4,581	1155833,307	4152505485039,225
$BRBFS^*$	9,404	1043,507	3285570,579	112529752937356,781
$IBRBFS^*$	0,187	0,049	66663,642	4758384417,347

Tabla 1: Observaciones del grafo del Metro de Madrid

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%, no sólo para la resolución del peor caso, sino también en término medio.

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*. Por otra parte, estos resultados fueron, también en término medio, muy parecidos a los obtenidos con el algoritmo RBFPS*. Nótese que los algoritmos de perímetro, a pesar de reducir considerablemente el tiempo necesario para encontrar la solución, ocupan una cantidad de memoria que es, en proporción, mayor que el ahorro de tiempo conseguido.

6.2 El juego del N-'Puzle'

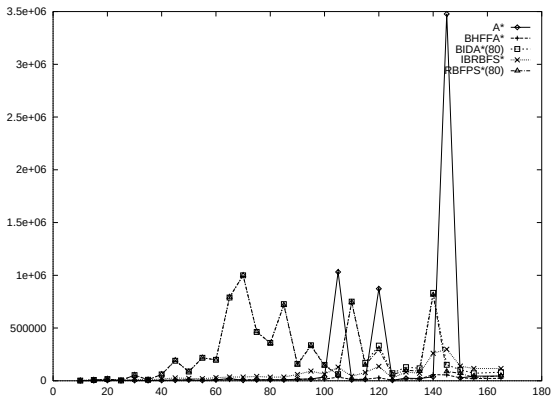
El juego del rompecabezas es, sin ningún lugar a dudas, el dominio más usado en los experimentos sobre búsqueda —del que existen innumerables referencias. El juego del N-'Puzle' consiste en un tablero de $N \times N$ posiciones, sobre el que se disponen $N^2 - 1$ fichas numeradas, de modo que se deja 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 colocar 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 $(16^2)!/2 = 10.461.394.944.000 \approx 10^{13}$ posiciones [3] y el factor de ramificación puro es 3. Además, se ha demostrado que el N-'Puzle' es un problema NP-duro [21].

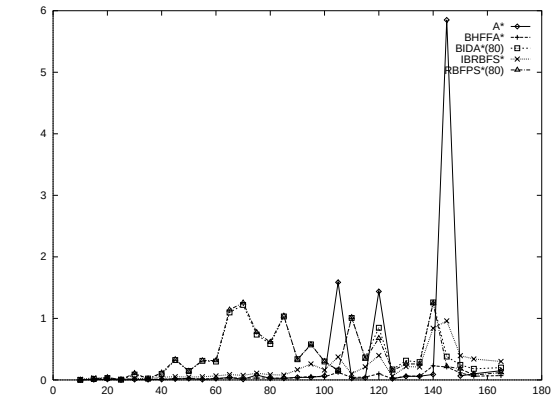
Este problema pertenece a la clase de los dominios *discretos*, porque el coste de cada arco es siempre el mismo e igual a la unidad, y esta es una cantidad natural o discreta. Por otra parte, la función heurística empleada es la *distancia Manhattan*, que es igual a la suma, para todas las fichas del 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 final o meta.

El histograma de la figura 8 muestra las frecuencias relativas de las distancias heurísticas entre 50 pares de configuraciones escogidas aleatoriamente.

Como es bien sabido, el principal inconveniente



(a) Memoria consumida (en bytes)



(b) Tiempo consumido (en segundos)

Figura 7: Mediciones del Metro de Madrid

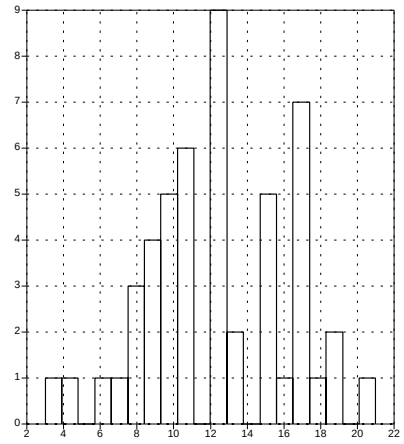


Figura 8: Histograma de distancias heurísticas del juego del 15-'Puzle'

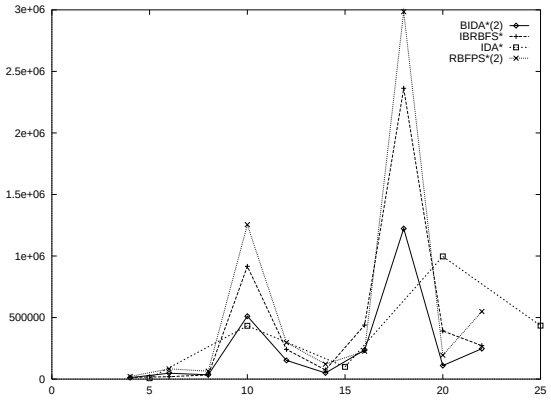
del algoritmo A* es, como el de todos los algoritmos de el mejor primero (salvo el RBFS), su alto consumo de memoria. Debido a ello, el A* no resolvió ni siquiera la mitad de los problemas propuestos, por lo que no es posible ofrecer resultados de su ejecución. Sin embargo, el BHFFA* sí los resolvió todos. La figura 9 muestra, gráficamente, el consumo total de memoria y tiempo para la resolución de todos los casos con los mejores algoritmos, en función de la diferencia heurística entre pares de nodos.

La tabla 2 muestra los recursos computacionales consumidos, en término medio, por cada algoritmo implementado, junto con las varianzas de cada observación. De dicha tabla se han excluído: el algoritmo RBFS porque Korf [13] ya ha realizado una cantidad exhaustiva de pruebas en este mismo dominio; el BRBFS*, porque el algoritmo IBRBFS* siempre será, naturalmente, mejor que él (como de hecho ocurrió en el dominio continuo)

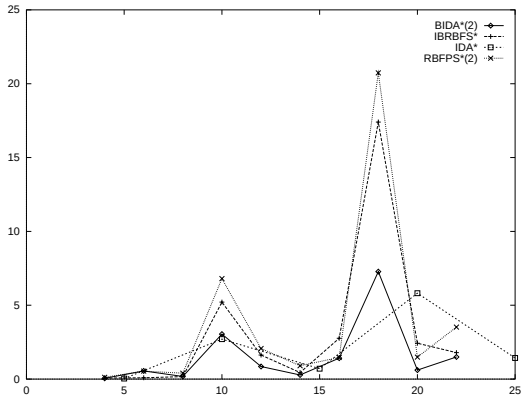
En este caso, el BHFFA* no fue, ni mucho menos, el mejor algoritmo. El IDA* consumió, aproximadamente, 20 veces menos recursos computaciones que él. Además, mientras que el tiempo medio para la resolución de las 50 instancias fue de 52,23 segundos, todos los perímetros probados con los algoritmos BIDA* y RBFPS* (2, 4 y 6) ofrecieron mejores resultados, tal y como se muestra en la tabla 2. Nótese que, en este caso, el algoritmo BIDA* fue superior al RBFPS* (al contrario que en el grafo del Metro de Madrid). Esto se debe a que, en este dominio, el algoritmo IDA* es más eficiente que el RBFS.

Por otra parte, aunque el algoritmo IBRBFS* fue mejor que el RBFPS* (para cualquiera de los perímetros probados), su consumo de memoria y tiempo fue superior que los del BIDA* y el IDA*.

Por otra parte, mientras que Korf [13] documentó que el algoritmo RBFS genera en este dominio un 1,4% menos nodos que el IDA*, se observó que el algoritmo IBRBFS* (que es una implementación bidireccional del RBFS) genera un 59,4% menos nodos que el algoritmo IDA*. La diferencia, sustancial, sirve para justificar por qué, el algoritmo, IDA* consume una cantidad de memoria que es, en proporción, mayor que la consumida por el IBRBFS*, en relación con el tiempo ahorrado por el primer algoritmo



(a) Memoria consumida (en bytes)



(b) Tiempo consumido (en segundos)

Figura 9: Mediciones del 15-'Puzle'

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
$RBFPS^*_2$	3,804	35,38	579720,405	763315379734,407
$RBFPS^*_4$	4,973	48,079	726649,924	937344862100,651
$RBFPS^*_6$	9,281	63,43	1363497,26	1156896259701,904
IBRBFS*	3,199	24,721	476345,607	463649693517,31

Tabla 2: Observaciones del 15-'Puzle'

(la razón del tiempo consumido por el algoritmo IDA* y el algoritmo IBRBFS* es, aproximadamente, 1,5, mientras que la razón del consumo es solo de 1,2).

7 Conclusiones

En primer lugar, se destaca el pésimo resultado del algoritmo IDA* en el escenario continuo. Aunque se trata de una idea sencilla y muy eficiente para el caso discreto —y por lo tanto, una idea genial—, su resultado es muy pobre en el caso continuo y, por ello, *cuando los costes de los arcos sean muy diferentes* (lo que ocurre con facilidad en el caso continuo), *se desaconseja, plenamente, la utilización del IDA**. Muy al contrario que en el caso en que los costes sean muy parecidos entre ellos —y tanto mejor si son iguales— en cuyo caso, este algoritmo puede ofrecer excelentes resultados.

Otra observación de importancia es que el algoritmo BHFFA* puede no dar tan excelentes resultados, cuando el número de nodos crece en exceso (como pasa en el 15-"Puzle"), puesto que el número de productos cruzados entre las fronteras crece desproporcionadamente y, con ello, el número de evaluaciones heurísticas (probablemente, una de las partes más costosas). En tal caso, *cuando el número de nodos crece, el algoritmo IBRBFS* resulta más apropiado* (en el grafo del Metro, su media fue de 0,14 segundos, contra los 0,391 del $RBFPS^*_{80}$), *a no ser que los costes de los arcos sean muy parecidos, en cuyo caso, lo mejor será el BIDA**. En este último caso, siempre será necesario realizar un estudio del problema a resolver para encontrar el tamaño de perímetro más apropiado.

Asimismo, una cuestión de muchísima relevan-

cia es, de hecho, el tiempo que consume la evaluación heurística. *Cuando el tiempo de evaluación es alto, debería dársele preferencia al algoritmo IDA* y, en otro caso, al algoritmo IBRBFS** quien expande y genera muchísimos menos nodos que el anterior (mientras que en el dominio discreto el ahorro fue del 59,4%, en el dominio continuo el algoritmo IBRBFS* sólo generó un 16,77% de los nodos generados por el IDA*).

Por último, *cuando ocurra que no es posible disponer de una heurística consistente, las únicas alternativas —a fin de garantizar la optimalidad de las soluciones obtenidas— son el RBFS y el RBFPS**. En los dominios estudiados, siempre se han encontrado valores de perímetro para mejorar los resultados de la versión unidireccional, por lo que se recomienda, nuevamente, el algoritmo bidireccional pero, se advierte, que es necesario estudiar el dominio para encontrar el tamaño de perímetro óptimo.

En cualquier caso, se ha observado que *los mejores algoritmos de búsqueda han sido siempre los algoritmos de búsqueda bidireccionales*:

- En el dominio continuo, el algoritmo BHFFA* ha sido, de todos, el mejor. De hecho, su diferencia con otros algoritmos, como el A* o el BIDA* es muy significativa.
- En el dominio discreto, el mejor algoritmo fue el BIDA^*_2 superando, ligeramente, los resultados obtenidos por el algoritmo IDA*.
- En cualquier escenario, el algoritmo IBRBFS* mejoró al RBFS, el algoritmo BIDA* al IDA* y, por último, el BHFFA*, al A*.

Considerando además que Hermann Kaindl y Gerhard Kainz [10] llegaron a conclusiones parecidas en otra categoría de algoritmos bidireccionales y que, asimismo, Toru Ishida logró mejorar también un algoritmo unidireccional de búsqueda de objetivos en movimiento [8] con una implementación bidireccional [7], se anima a la comunidad investigadora a que desarrolle algoritmos de búsqueda bidireccional a partir de las ventajas logradas con un nuevo algoritmo de búsqueda unidireccional.

8 Líneas futuras

A la vista del estado actual de la investigación en este área de la Inteligencia Artificial, y de los resultados obtenidos, se plantean —razonadamente— los siguientes puntos de desarrollo e investigación:

- Los planteados por Manzini [16]:
  1. Investigación del análisis de técnicas de evaluación perezosa que sirvan para minimizar, tanto como sea posible, el número de evaluaciones heurísticas, de modo que pueda mejorarse aún más el consumo de los recursos computaciones como el tiempo y la memoria.
  2. El análisis de algoritmos de perímetro, que empleen funciones heurísticas definidas sobre conjuntos que, como en el caso del BRBFS* e IBRBFS*, sean dinámicos, esto es, que puedan variar durante la búsqueda.
  3. Estudio de la implementación paralela de los algoritmos de búsqueda bidireccionales, por ejemplo, los presentados en este trabajo.
- En consideración con el algoritmo BHFFA*, indagar si es posible, o no, superar el inconveniente de alternar entre los sentidos de la búsqueda, para que también se puedan aplicar sobre él, principios como el de cardinalidad de Pohl.
- Realizar experimentos adicionales sobre dominios continuos con distancias poco variables y sobre dominios discretos con distancias muy variables, para reforzar los resultados presentados en este trabajo.
- Estudio de la generalización de los algoritmos de búsqueda presentados en este trabajo, para que puedan implementarse en una sola arquitectura, tal y como sucede en los algoritmos de búsqueda de dos agentes, con MTD [19].

Referencias

[1] Dennis De Champeaux. Bidirectional heuristic search again. *Journal of the Associa-*

tion for Computing Machinery, 30(1):22–32, January 1983.

[2] Dennis De Champeaux and Lenie Sint. An improved bidirectional heuristic search algorithm. *Journal of the Association for Computing Machinery*, 24(2):177–191, April 1977.

[3] Joseph C. Culberson and Jonathan Schaeffer. Efficiently searching the 15-puzzle. Technical Report TR 94-08, University of Alberta, may 1994.

[4] John F. Dillenburg and Peter C. Nelson. Perimeter search. *Artificial Intelligence*, 65:165–178, 1994.

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

[6] Sixto Ríos Insua. *Investigación Operativa*. Centro de Estudios Ramón Areces, Madrid, 1988.

[7] Toru Ishida. Real-time bidirectional search: Coordinated problem solving in uncertain situations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):617–627, June 1996.

[8] Toru Ishida and Richard E. Korf. Moving-target search: A real-time search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):609–619, June 1995.

[9] Guy L. Steele Jr. *Common Lisp: The Language*. Digital Press, second edition, 1990.

[10] Hermann Kaindl and Gerhard Kainz. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, 7:283–317, December 1997.

[11] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[12] Richard E. Korf. Linear-space best-first search: Summary of results. In *Proceedings AAAI-92*, pages 533–538, 1992.

[13] Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[14] Carlos Linares. Algoritmos de búsqueda de un agente en dominios discretos y continuos. Tesis de Master, Facultad de Informática. Universidad Politécnica de Madrid, Mayo 1997.

[15] Carlos Linares and Asunción Gómez. BHFFA*: Un nuevo algoritmo admisible de búsqueda bidireccional. In *Séptima Conferencia de la Asociación Española para la Inteligencia Artificial*, Málaga (España), Noviembre 1997. AEPIA.

[16] Giovanni Manzini. BIDA*: an improved perimeter search algorithm. *Artificial Intelligence*, 75:347–360, 1995.

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

[18] Judea Pearl. *Heuristics*. Addison-Wesley, Reading MA, 1984.

[19] Aske Plaat. *Research Re: Search & Re-Search*. PhD thesis, Rotterdam, March 1996.

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

[21] Daniel Ratner and Manfred Warmuth. Finding a shortest solution for the n×n extension of the 15-puzzle is intractable. In *Proceedings AAAI-86*, pages 168–172, 1986.