Resolución de problemas de búsqueda.

Memoria de prácticas de Inteligencia Artificial.

Primera entrega.

Francisco Abel Cedrón Santaeufemia francisco.cedron
Pablo García Vila pablo.gvila

Resumen.

La práctica trata la búsqueda de los movimientos necesarios para que un caballo de ajedrez se desplace de una casilla origen de un tablero, de dimensiones dadas, a una de destino.
Con el fin de resolver el problema que nos atañe se emplearán los algoritmos de búsqueda vistas en la parte teórica de la asignatura:

- Búsqueda ciega: en anchura y en profundidad
- Búsqueda A*: implementará funciones heurísticas definidas por los autores de la práctica.

En esta parte analizaremos dichos métodos, lo que nos llevará a una mejor comprensión de los mismos, y trataremos de sentar unas bases para la posterior implementación de una solución al problema dado.

Tabla de contenidos.

Métodos.

Conceptos previos.

Un problema puede definirse por cuatro componentes:

- El **estado inicial** en el que comienza el agente[1].
- Una descripción de las posibles acciones disponibles por el agente. Dado un estado particular *x*, la **función sucesor** SUCESOR-FN(x) devuelve un conjunto de pares ordenados <*acción*, *sucesor*>, donde cada acción es una de las acciones legales en el estado *x* y cada sucesor es un estado que puede alcanzarse desde *x*, aplicando la acción.
- Implícitamente el estado inicial y la función sucesor definen el espacio de estados del problema que es el conjunto de todos los estados alcanzables desde el estado inicial. El espacio de estados forma un grafo en el cual los nodos son estados y los arcos entre estados son las acciones.
- La **prueba de meta**, la cual determina si un estado es objetivo[2]. Algunas veces existe un conjunto explicito de posibles estados objetivo y el test simplemente comprueba si el estado es uno de ellos.
- Una **función de coste** que asigna un coste numérico a cada camino. El agente resolvente de problemas elige una función de coste que refleje nuestra medida de rendimiento. El coste individual de una acción a que va desde un estado x al estado y se denota por $c(x, a, y)$.

Los elementos anteriores definen un problema y pueden unirse en una estructura de datos simple que se dará como entrada al algoritmo resolvente del problema. Una solución de un problema es un camino desde el estado inicial a un estado objetivo. La calidad de la solución se mide por la función de coste del camino, y una solución óptima tiene el coste más pequeño del camino entre todas las soluciones.
Aplicando las definiciones anteriores al problema en cuestión tenemos:
Espacio de estados. Definido por un par de valores x, y de la siguiente manera:

$$(x, y)\ tal\ que\ x, y \in [0,9]$$

Estado inicial. Cualquier par de valores (x, y) definido en el conjunto de espacio de estados donde se encuentra el caballo en el tablero.
Estado meta. Cualquier par de valores (x, y) definido en el conjunto de espacio de estados donde se quiere que llegue el caballo.
Lista de operadores. Todos los posibles movimientos que se pueden realizar con el caballo, los cuales están definidos en la siguiente tabla.

[1] Un agente es cualquier cosa capa de percibir el entorno que lo rodea con la ayuda de sensores y actuar en ese medio utilizando actuadores.
[2]El estado objetivo de un problema también es conocido como estado meta.

Tabla 1. Lista de operadores.

Operador	Resultado del operador	Condiciones
Op0(x, y)	(x-1, y-2)	x ∈ [1, 9], y ∈ [2, 9]
Op1(x, y)	(x+1, y-2)	x ∈ [0, 8], y ∈ [2, 9]
Op2(x, y)	(x+2, y-1)	x ∈ [0, 7], y ∈ [1, 9]
Op3(x, y)	(x+2, y+1)	x ∈ [0, 7], y ∈ [0, 8]
Op4(x, y)	(x+1, y+2)	x ∈ [0, 8], y ∈ [0, 7]
Op5(x, y)	(x-1, y+2)	x ∈ [1, 9], y ∈ [0, 7]
Op6(x, y)	(x-2, y+1)	x ∈ [2, 9], y ∈ [0, 8]
Op7(x, y)	(x-2, y-1)	x ∈ [2, 9], y ∈ [1, 9]

Prueba de meta. Comprobar que el estado actual en el que se encuentra el caballo es el mismo que el estado meta definido anteriormente, en otras palabras sería lo mismo que decir que las coordenadas *x* e *y* del estado actual coincidieran con las coordenadas *x* e *y* de la posición que se desea alcanzar.

Función de coste. Como todos los operadores se mueven el mismo número de casillas vamos a definir que la función de coste sea la siguiente para todas las casillas:

$$\int_{coste}(x) = 1 \quad x \in Listadeoperadores$$

Análisis de los métodos de búsqueda ciega.

En este apartado se explicarán en qué consiste los algoritmos de búsqueda ciega: primero en anchura y primero en profundidad. Al final de la sección en la tabla 2 se muestra una comparativa de los requerimientos de cada algoritmo.

Búsqueda primero en anchura.

La búsqueda primero en anchura es una estrategia sencilla en la que se expande primero el nodo raíz, a continuación se expanden todos los sucesores del nodo raíz, después sus sucesores, etc. En general se expanden todos los nodos a una profundidad en el árbol de búsqueda antes de expandir cualquier nodo del próximo nivel.

La búsqueda primero en anchura se puede implementar con una frontera vacía que sea una cola FIFO, asegurando que los nodos primeros visitados serán los primeros expandidos. La cola FIFO pone todos los nuevos sucesores generados al final de la cola, lo que significa que los nodos más superficiales se expanden antes que los nodos más profundos. La figura 1 muestra un ejemplo del proceso, donde el nodo apuntado es el siguiente a expandir y los nodos con fondo son los que se hallan en la frontera.

Podemos ver que la búsqueda primero en anchura es siempre completa[3], siempre que el nodo que contenga el estado objetivo este en una profundidad finita y el factor de ramificación b también sea finito. El nodo objetivo más superficial no tiene porque ser el nodo óptimo.

Para poder analizar los requerimientos de memoria y tiempo del proceso de búsqueda analizaremos un problema hipotético, donde cada estado tiene b sucesores. El nodo raíz del árbol de búsqueda genera b nodos para el primer nivel, cada uno de ellos genera b nodos, con lo que el segundo nivel tenemos b^2 nodos, y así sucesivamente para los siguiente niveles. Ahora vamos a suponer que la solución está en una profundidad d. En el peor de los casos se expandirán todos los nodos a excepción de el último nodo en el nivel d(ya que el nodo que contiene el estado meta no se expande), generando $b^{d+1} - b$ nodos en el nivel $d + 1$. Así podemos concluir que el número de nodos generados es el siguiente:

$$b + b^2 + \cdots + (b^{d+1} - b) = \partial(b^{d+1})$$

Como cada nodo generado puede ser un posible nodo que pertenezca al camino de la solución tiene que permanecer en memoria. Por lo tanto, la complejidad en espacio es la misma que la complejidad en tiempo.

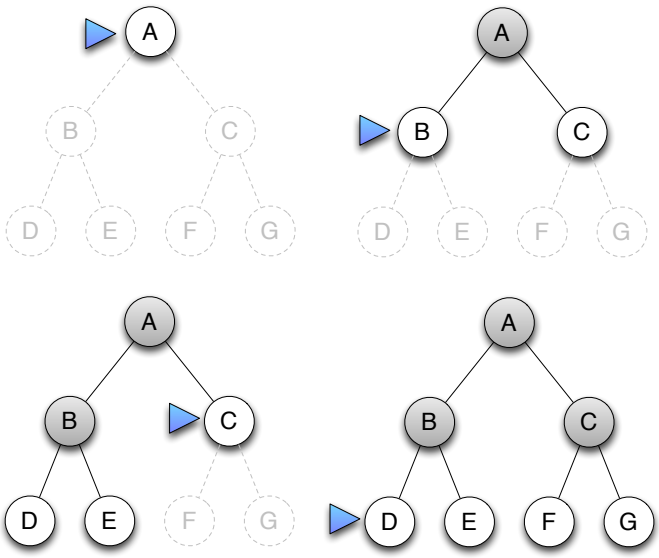

Figura 1. Ejemplo de búsqueda primero en anchura.

Búsqueda primero en profundidad.

La búsqueda primero en profundidad siempre expande el nodo más profundo en la frontera actual del árbol de búsqueda. La búsqueda procede inmediatamente al nivel más profundo del árbol de búsqueda, donde los nodos no tienen ningún sucesor. Cuando esos nodos se expanden, son quitados de la frontera, así entonces

[3]Que una búsqueda sea completa es lo mismo que decir que el proceso de búsqueda es capa de hallar una solución.

la búsqueda "retrocede"[4] al siguiente nodo más superficial que todavía no tenga sucesores inexplorados.
Esta estrategia puede implementarse usando una cola LIFO para la frontera.
La búsqueda primero en profundidad tiene unos requisitos modestos de memoria. Sólo se necesita almacenar un camino desde la raíz a un nodo hoja junto con los nodos restantes no expandidos para cada nodo del camino. Una vez que un nodo ha sido expandido, se puede quitar de la memoria tan pronto como todos sus descendientes han sido explorados. Para un espacio de estados con factor de ramificación *b* y máxima profundidad *m*, la búsqueda primero en profundidad requiere almacenar sólo *bm +1* nodos. En la figura 2 se puede ver un ejemplo del proceso de búsqueda primero en profundidad donde los que han sido expandidos y no tienen descendientes en la frontera se eliminaron de la memoria y aparecen en negro, el nodo que va a ser expandido aparece marcada con un triángulo y el nodo M es el que contiene el estado objetivo.

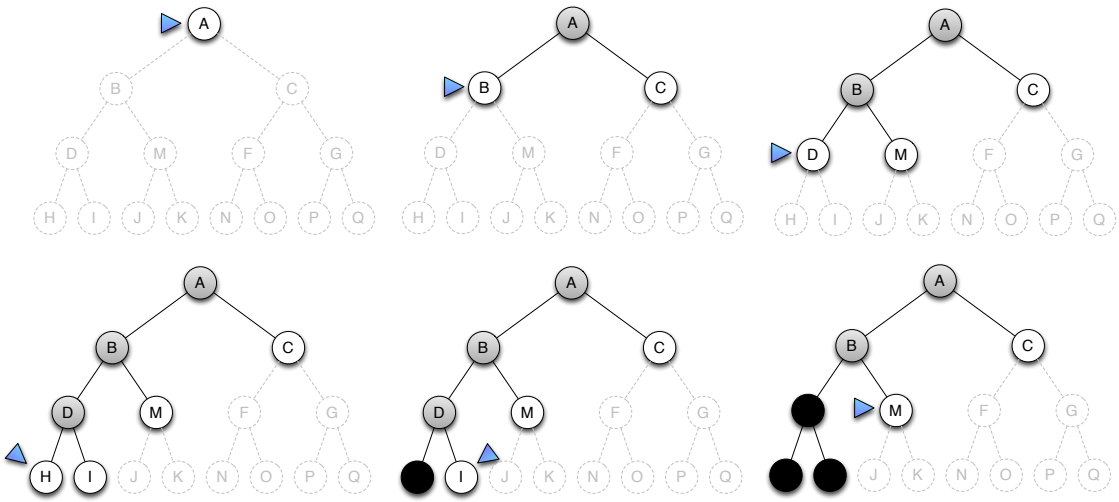

Figura2. Ejemplo de búsqueda primero en anchura.

Tabla 2. Comparación de las estrategias de búsqueda.

	¿Completa?	**Tiempo**	**Espacio**	**¿Óptima?**
Primero en anchura	Sí[a]	$\partial(b^{d+1})$	$\partial(b^{d+1})$	Sí[b]
Primero en profundidad	No	$\partial(b^{m})$	$\partial(bm)$	No
b es el factor de ramificación; *d* es la profundidad de la solución más superficial; *m* es la máxima profundidad del árbol. Los superíndices significan lo siguiente: a: Completa si *b* es finita. b: Completa si los costes para los todos los operadores el coste es ε (para ε> 0).				

[4]El proceso que hace que la búsqueda "retroceda" se conoce como backtracking.

Concepto de función heurística.

El problema de las búsquedas ciegas es que no tienen conocimiento más allá del nodo que utilizan para expandir lo que provoca en la búsqueda primero en profundidad una expansión masiva de nodos (afectando esto al gasto de memoria y a la cantidad de tiempo empleada para hallar la solución) mientras que, en la búsqueda en profundidad se puede quedar "atrapada" en una rama infinita (haciendo que no se encuentre nunca la solución). En este párrafo mostraremos como utilizando conocimiento específico del problema se pueden encontrar soluciones de manera más eficiente.

La primera aproximación que se puede encontrar de búsqueda informada es la búsqueda de primero el mejor, en donde a la hora de expandir se escoge el nodo en base a una función de evaluación.

Existe una gran familia de algoritmos de búsqueda primero el mejor con distintas funciones de evaluación, pero una componente clave de estos algoritmos es la función heurística denotada h(n). El objetivo de la función heurística es el de poder cuantificar la estimación del coste desde un estado hasta el estado meta.

En concreto para el problema propuesto en la práctica se definen tres heurísticas diferentes:

Primera heurística.

La función heurística escogida como primera función heurística es la que siempre devuelve cero.

$$h_1(n) = 0$$

Con esta función heurística, nunca se sobreestima el coste en un estado. Es una mala heurística a emplear, dado a que no revela ninguna información acerca de cuán lejos está un estado del objetivo que se quiere alcanzar. La implementación de esta heurística solo se empleará porque usada en el algoritmo de búsqueda A*, tiene el mismo comportamiento que la búsqueda de costo uniforme.

Esta heurística no sobreestima nunca el coste real debido a que siempre devuelve cero, la cual es el mismo razonamiento para explicar que no existen mínimos locales en la función.

Segunda heurística.

Esta segunda heurística se basa en la idea de cómo se mueve el caballo. Como su movimiento es en forma de L, se puede entender que primero se desplaza dos casillas en una dirección y después una en otra. Esta manera de pensar el movimiento nos sirve para indicar que el caballo en total se mueve tres casillas por desplazamiento, haciendo que nunca se tenga que mover en diagonal, con lo que a la hora de estimar la heurística se cuenta el número de casillas que se tiene que mover en una dirección y a continuación se le suman las casillas que le restan en la otra dirección.

$$h_2(n) = ceil\left(\frac{|x_i - x_m| + |y_i - y_m|}{3}\right)$$

En esta heurística nunca se sobreestima, porque no se tiene en cuenta el número de casillas que se mueve el caballo. En la figura 3, se muestra un tablero, para ver como aumentan los costes con referencia que se aleja de la casilla de inicio.

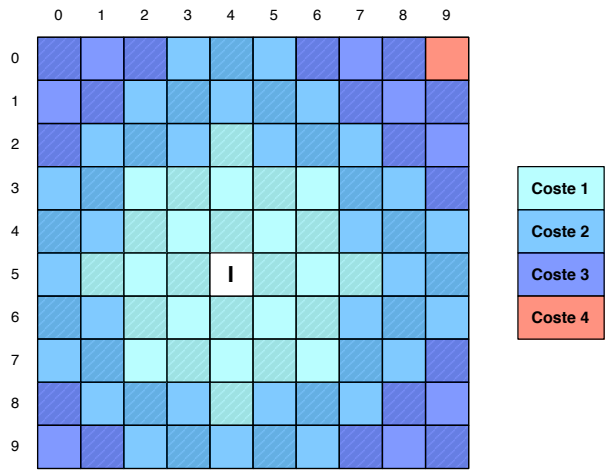

Figura 3. Idea en la que se basa la heurística 2.

Tercera heurística.

Esta función heurística se basa en la longitud que se desplaza el caballo, con lo que obtenemos que lo máximo que se aleja es de dos casillas en una de sus coordenadas. La idea es dividir el tablero en cuadrados, donde el primer cuadrado lo conforma la casilla donde está el caballo y los siguientes distan dos casillas del anterior. Con esto tenemos la siguiente ecuación donde (x_m, y_m) es el estado meta a alcanzar y el estado (x_i, y_i) es el estado donde se encuentra el caballo, y las divisiones dan como resultado un número entero:

$$h_3(n) = \begin{cases} 0 & si\ x_i = x_m \wedge y_i = y_m \\ max\left(\frac{||x_i - x_m| - 1|}{2} + 1, \frac{||y_i - y_m| - 1|}{2} + 1\right) & en\ otro\ caso \end{cases}$$

Como en esta heurística lo que se hace es contar el número de cuadrados que separan el estado actual del estado meta y ese número la cantidad mínima para llegar al estado objetivo nunca va a sobreestimar el número de movimientos.

En cuanto a si la heurística presenta mínimos locales, por la explicación que se ha dado anteriormente se responde que no, pero si en la función se cambia el operador de selección de máximo por un mínimo, existirían mínimos locales, porque contaría la mínima longitud que hay que "recorrer" para que una de las coordenadas esté en la misma posición que una de las que hay en el estado objetivo.

La siguiente figura muestra como la idea de la heurística en la que se muestra un tablero de 10x10, y los recuadros azules muestran del más interior al más exterior el número mínimo de movimientos si los empezamos a contar desde cero. Para el ejemplo de la figura el estado inicial está marcado con una I y el estado meta con una *M*, y si empezamos a contar está en el 3 cuadrado que se corresponde con el número mínimo de 2 desplazamientos para alcanzar ese cuadrado.

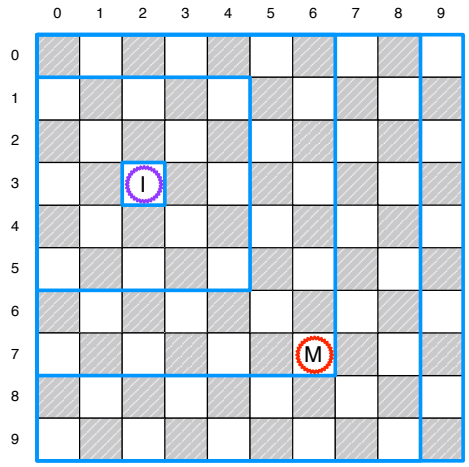

Figura 4. Idea en la que se basa la heurística 3.

Existencia de muros verticales.

En la situación particular de que existiesen muros verticales en el que el caballo no pudiese pasar por esa casilla explicaremos como se comportarían las heurísticas.
Como la existencia de muros verticales haría que el caballo tuviera que rodearlo, haciendo que así tuviera que recorrer más casillas en las que necesarias sin la existencia de los mismos, con lo que deducimos que ninguna de las heurísticas sobreestiman el coste. Lo único que se podría dar lugar en la segunda y tercera heurística es en la aparición de mínimos locales en el espacio de estados. La primera heurística al devolver siempre cero nunca generará mínimos locales en el espacio de estados.
Además para las diferentes estrategias se tendrá en cuenta si el algoritmo empleado usa un árbol o un grafo, en donde la única diferencia entre ellos es que usando un árbol se expanden todos los nodos, mientras que usando un grafo, nunca se expande un nodo que contenga un estado que ya haya sido expandido en un nodo con anterioridad. Así en la tabla 3 mostramos los posible resultados con las diferentes estrategias de búsqueda.

Tabla 3. Resultados posibles en las heurísticas con muros verticales.

	Primero en profundidad	**Primero en Anchura**	**A***
Árbol	No es completa[a].	Completa pero no óptima.	Completa pero no óptima.
Grafo	Completa pero no óptima[b].	Completa pero no óptima.	Completa pero no óptima.
a: No es completa porque se puede quedar explorando una rama con un bucle. b: Es completa porque al no expandirse ningún nodo con estados repetidos, puede llegar a explorar todas las casillas del tablero.			

La pieza es el rey.

En el caso particular de que la pieza fuese un rey en lugar de que sea un caballo, hay que analizar las heurísticas debido a que los operadores cambian teniendo una lista de operadores como la que se muestra en la tabla 4.

Tabla 4. Lista de operadores donde la pieza es un rey.

Operador	**Resultado del operador**	**Condiciones**
Op0(x, y)	(x+1, y)	$x \in [0, 8], y \in [0, 9]$
Op1(x, y)	(x+1, y-1)	$x \in [0, 8], y \in [1, 9]$
Op2(x, y)	(x, y-1)	$x \in [0, 9], y \in [1, 9]$
Op3(x, y)	(x-1, y-1)	$x \in [1, 9], y \in [1, 9]$
Op4(x, y)	(x-1, y)	$x \in [1, 9], y \in [0, 9]$
Op5(x, y)	(x-1, y+1)	$x \in [1, 9], y \in [0, 8]$
Op6(x, y)	(x, y+1)	$x \in [0, 9], y \in [0, 8]$
Op7(x, y)	(x+1, y+1)	$x \in [0, 8], y \in [0, 8]$

Primera heurística.

La función heurística escogida como primera función heurística es la que siempre devuelve cero por ese motivo nunca se sobreestima el coste real que hay para llegar por un camino óptimo desde el estado actual al estado meta. Ese es el motivo por el que sigue siendo válida y no se necesita adaptar .

Segunda heurística.

La aplicación de esta heurística al problema sería válida, si bien no óptima, ya que al tener el rey un alcance lineal menor, que el considerado para el caballo, jamás se sobreestimaría el coste de cualquier camino hasta un estado objetivo dado.

De todas maneras, no sería difícil de adaptarla para mejorarla de tal manera que en este caso habría que tener en cuenta la distancia euclídea porque el rey tiene en cuenta los movimientos diagonales.

$$h_2(n) = ceil\left(\sqrt{(x_i - x_m)^2 + (y_i - y_m)^2}\right)$$

Tercera heurística.

Como en esta heurística se tiene en cuenta la longitud máxima que alcanza el caballo en alguna de sus coordenadas, que es de dos casillas, y difiere con el rey, que es de una casilla, es necesaria adaptar la heurística para mejorar los costes de un camino óptimo hacia el estado objetivo.

Adaptando la heurística a los nuevos operadores que tiene el caballo sería la siguiente:

$$h_3(n) = \begin{cases} 0 & si\ x_i = x_m \wedge y_i = y_m \\ max(|x_i - x_m|, |y_i - y_m|) & en\ otro\ caso \end{cases}$$

Diferencia de coste entre operadores.

Teniendo en cuenta de que la pieza sigue siendo el rey, y que ahora algunos operadores tienen un incremento de coste en alguno de los operadores, el coste del camino óptimo puede variar, haciendo que este incremente, pero haciendo que nunca llegue a bajar con respecto al apartado anterior.

Como las heurísticas tenían en cuenta que el coste por operador es de una unidad, ahora al incrementarse el coste hace que la estimación del mismo no sea tan precisa. Aún así en ningún lugar llega a sobreestimar el coste en ninguna de las tres heurísticas propuestas en el apartado anterior.

Bibliografía.

[1] S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach.* Second Edition. Pretince Hall. 2002.

[2] R. Marín, J. Palma. *Inteligencia Artificial. Técnicas, métodos y aplicaciones.* Primera Edición. McGraw-Hill. 2008.

[3] V. Moret, A. Alonso, M. Cabrero, B. Guijarro, E. Mosqueira. *Fundamentos de Inteligencia Artificial.* Segunda Edición. Servicio de publicaciones UDC. 2000.