

Algoritmo de búsqueda A^*

Stalin Muñoz Gutiérrez

Centro de Ciencias de la Complejidad
Universidad Nacional Autónoma de México (UNAM)

Vamos ahora a ver el algoritmo denominado A^* .

El algoritmo A^* fue propuesto por Peter Hart, Nils Nilsson y Bertram Raphael en los años 60's.

Aún sigue siendo uno de los algoritmos más utilizados para encontrar rutas en grafos pesados.

Algoritmo A^*

Basado en UCS

El algoritmo A^* es como un algoritmo UCS, con dos cambios:

- 1 un parametro adicional que es la función heurística,
 $h : S \rightarrow \mathbb{R}$.
- 2 la cola de prioridad utiliza como criterio de ordenamiento la función $f : S \rightarrow \mathbb{R}$ definida como sigue:

$$f(n) = g(n) + h(n) \quad (1)$$

$g(n)$ es el costo acumulado del estado inicial al estado n .

Algoritmo de búsqueda A^*

└ Algoritmo A^*

Algoritmo A^*

Basado en UCS

El algoritmo A^* es como un algoritmo UCS, con dos cambios:

- un parametro adicional que es la función heurística,
 $h : S \rightarrow \mathbb{R}$.
- la cola de prioridad utiliza como criterio de ordenamiento la función $f : S \rightarrow \mathbb{R}$ definida como sigue:

$$f(n) = g(n) + h(n) \quad (1)$$

$g(n)$ es el costo acumulado del estado inicial al estado n .

A^* es una modificación del algoritmo UCS.

Tiene dos cambios importantes.

Primero, recibe una función heurística h , que estima el costo de un estado al estado meta.

Segundo, el criterio de ordenamiento para la agenda, ahora es la suma del costo acumulado desde el estado inicial, calculado con la función $g(n)$ y la estimación del costo al estado meta, obtenida con la heurística h .

Si $h(n)$ es igual a zero para todo estado n , recuperamos el algoritmo UCS.

Ejemplo: juego del 15

4	1	3
	2	5
7	8	6

Inicial

1	2	3
4	5	6
7	8	

meta

$h(n) = \#$ de piezas fuera de lugar

Ejemplo: juego del 15

4	1	3
2	5	
7	8	6

Inicial

1	2	3
4	5	6
7	8	

meta

$h(n) = \#$ de piezas fuera de lugar

Vamos a ilustrar el funcionamiento del algoritmo con el juego del 15.
Como función heurística usaremos el número de fichas fuera de lugar.

La meta es el arreglo ordenado de los números de menor a mayor.

Ejemplo: juego del 15

Agenda

4	1	3
	2	5
7	8	6

 A Algoritmo de búsqueda A^*

└ Ejemplo: juego del 15

A^* esta basado en UCS, usaremos una cola de prioridad como agenda.

Metemos el estado inicial a la agenda.

Agenda

 A

Ejemplo: juego del 15

Expandidos

 $[A]$

Agenda



A

sacamos a A de la agenda



Ejemplo: juego del 15

Lo sacamos de la agenda.

Agregamos a la lista de expandidos.

También es posible que eliminemos el conjunto de expandidos. En ese caso el algoritmo trabajará más pero usará menos memoria.

Ejemplo: juego del 15

Expandidos

$[A]$

Agenda



A



B_A^{1+4} C_A^{1+5} D_A^{1+6}

$$g(n) = 1$$

$$h(B) = 4$$

$$h(C) = 5$$

$$h(D) = 6$$

Algoritmo de búsqueda A^*

2018-10-02

Ejemplo: juego del 15

Ejemplo: juego del 15



Al expandir al estado A , Obtenemos 3 configuraciones B , C y D .

Aquí en amarillo mostramos las fichas fuera de lugar.

Como superíndice indicamos la función f para cada estado.

En el caso de B tenemos una $h(B) = 4$.

En todos los estados el costo acumulado es 1.

Ejemplo: juego del 15

Expandidos

$[A \ B]$

Agenda



A



B_A^{1+4}



C_A^{1+5}



D_A^{1+6}



E_B^{2+3}



C_A^{1+5}



D_A^{1+6}

Ejemplo: juego del 15



Al frente de la cola de prioridad queda B con un costo total de 5, los otros estados tienen costos mayores.

Expandimos a B , obteniendo el estado E .

Ejemplo: juego del 15

Expandidos

$[A \ B \ E]$

Agenda



A



B_A^{1+4}



C_A^{1+5}



D_A^{1+6}



E_B^{2+3}



C_A^{1+5}



D_A^{1+6}



F_E^{3+2}



G_E^{3+4}



C_A^{1+5}

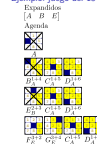


D_A^{1+6}

Algoritmo de búsqueda A^*

└ Ejemplo: juego del 15

Ejemplo: juego del 15



E resulta ser el de menor costo.

Lo expandimos generando dos sucesores: F y G .

Ejemplo: juego del 15

Expandidos

$[A \ B \ E \ F]$

Agenda

~~A~~
 $B_A^{1+4} \ C_A^{1+5} \ D_A^{1+6}$



$E_B^{2+3} \ C_A^{1+5} \ D_A^{1+6}$



$F_E^{3+2} \ G_E^{3+4} \ C_A^{1+5} \ D_A^{1+6}$



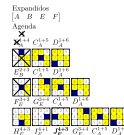
$H_F^{4+3} \ I_F^{4+1} \ J_F^{4+3} \ G_E^{3+4} \ C_A^{1+5} \ D_A^{1+6}$

Algoritmo de búsqueda A^*

2018-10-02

└ Ejemplo: juego del 15

Ejemplo: juego del 15



El siguiente estado en salir es F con un costo acumulado de 3 y un costo estimado a la meta de 2.

F generará 3 sucesores. El costo acumulado en todos ellos es de 4.

Hasta el momento el algoritmo no ha retrocedido.

Sigue avanzando en dirección a la meta.

Ejemplo: juego del 15

Expandidos

$[A \ B \ E \ F \ I]$

Agenda

~~A~~

~~B_A^{1+4}~~ C_A^{1+5} D_A^{1+6}

~~B_B^{2+3}~~ C_A^{1+5} D_A^{1+6}

~~B_E^{3+2}~~ G_E^{3+4} C_A^{1+5} D_A^{1+6}



H_F^{4+3} I_F^{4+1} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}



K_I^{5+0} L_I^{5+2} H_F^{4+3} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}

Algoritmo de búsqueda A^*

2018-10-02

└ Ejemplo: juego del 15

Ejemplo: juego del 15

Expandidos
 $[A \ B \ E \ F \ I]$
 Agenda
 ~~A~~
 ~~B_A^{1+4}~~ C_A^{1+5} D_A^{1+6}
 ~~B_B^{2+3}~~ C_A^{1+5} D_A^{1+6}
 ~~B_E^{3+2}~~ G_E^{3+4} C_A^{1+5} D_A^{1+6}
 H_F^{4+3} I_F^{4+1} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}
 K_I^{5+0} L_I^{5+2} H_F^{4+3} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}

Ahora al frente se encuentra el estado I .

Al expandirlo genera los estados K y L .

K es la meta.

Como en UCS, no hemos terminado, se agrega a la agenda.

Terminaremos cuando se encuentre al frente de la cola de prioridad.

Nos daremos cuenta al sacarlo.

Ejemplo: juego del 15

Expandidos

$[A \ B \ E \ F \ I]$

Agenda

~~A~~

~~B_A^{1+4}~~ C_A^{1+5} D_A^{1+6}

~~E_B^{2+3}~~ C_A^{1+5} D_A^{1+6}

~~E_E^{3+2}~~ G_E^{3+4} C_A^{1+5} D_A^{1+6}



H_F^{4+3} I_F^{4+1} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}



K_I^{5+0} L_I^{5+2} H_F^{4+3} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}

meta al frente
de la agenda

Algoritmo de búsqueda A^*

2018-10-02

└ Ejemplo: juego del 15

Sacamos el estado de menor costo f .

Ahora si, dado que la meta es el estado de menor costo, estamos listos para recuperar la ruta y finalizar la ejecución del algoritmo.

Ejemplo: juego del 15



Ejemplo: juego del 15

Expandidos

$[A \ B \ E \ F \ I]$

Agenda ruta = $[A \ B \ E \ F \ I \ K]$

~~A~~

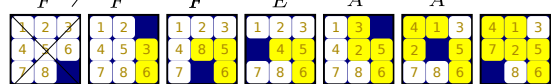
~~B~~_A¹⁺⁴ C_A^{1+5} D_A^{1+6}

~~E~~_B²⁺³ C_A^{1+5} D_A^{1+6}

~~E~~_E³⁺² G_E^{3+4} C_A^{1+5} D_A^{1+6}



H_F^{4+3} / I_F^{4+1} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}



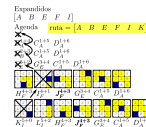
K_I^{5+0} L_I^{5+2} H_F^{4+3} J_F^{4+3} G_E^{3+4} C_A^{1+5} D_A^{1+6}

Algoritmo de búsqueda A^*

2018-10-02

└ Ejemplo: juego del 15

Ejemplo: juego del 15



Regresamos con los apuntadores a los estados predecesores.

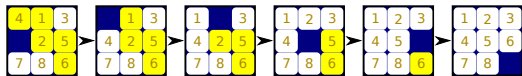
La ruta esta dada por la secuencia A,B,E,F,I,K, para simplificar hemos omitido la anotar la acción necesaria para lograr cada estado.

Ejemplo: juego del 15

Expandidos

 $[A \ B \ E \ F \ I]$

Agenda

ruta = $[A \ B \ E \ F \ I \ K]$ 

Ejemplo: juego del 15

Expandidos
 $[A \ B \ E \ F \ I]$
 Agenda ruta = $[A \ B \ E \ F \ I \ K]$

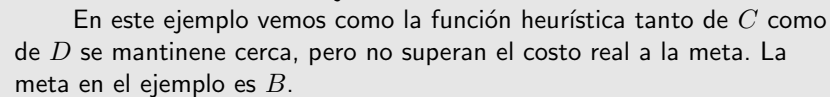
Aquí la secuencia de movimientos que nos llevan a la meta.

La heurística guió al algoritmo en dirección a la meta. Llegamos muy rápido a la solución.

El éxito de la aplicación de A^* depende mucho de la función heurística seleccionada.

└ Admisibilidad en A^*

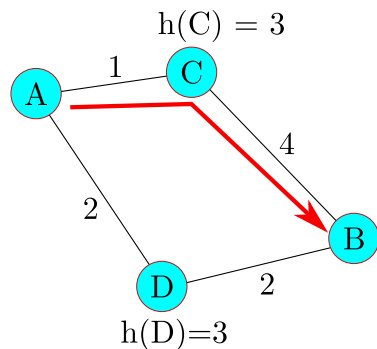
La admisibilidad de una heurística consiste en que el valor de la función heurística para todo nodo de la red debe ser inferior o igual al valor mínimo de costo al objetivo.



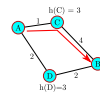
Si calculamos el valor de f para C y D observamos que es de 4 y 3 respectivamente. Por lo tanto el agente prefiere tomar la ruta por D .

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

Admisibilidad



Con una heurística no admisible, el agente puede tomar una ruta más larga.



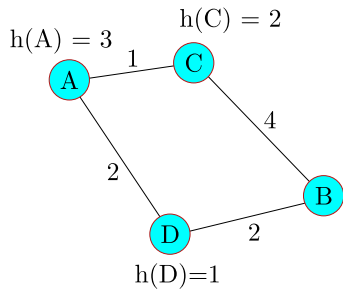
Con una heurística no admisible, el agente puede tomar una ruta más larga.

En este ejemplo tenemos una heurística no admisible.

Pues está sobreestimando el costo para el nodo D , con ello conduciendo al agente por un camino más largo.

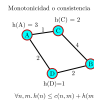
Consistencia en A^*

Monotonidad o consistencia



$$\forall n, m. h(n) \leq c(n, m) + h(m)$$

2018-10-02

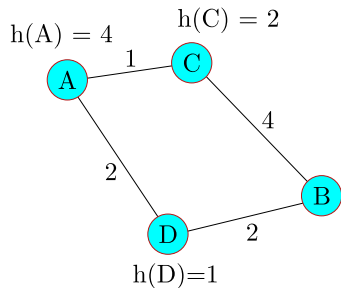
└ Consistencia en A^* 

Una heurística es consistente o monotónica si para cada sucesor de todo vertice del grafo de estados acciones, la heurística del vértice es menor o igual que la heurística de la suma del costo de la transición del vertice al sucesor con la heurística del sucesor.

En este ejemplo la heurística mostrada cumple con esta propiedad.

Consistencia en A^*

Monotonidad o consistencia



No monotónica

Algoritmo de búsqueda A^* └ Consistencia en A^* Consistencia en A^* 

Aquí ilustramos un ejemplo donde la heurística no es monotonía.

Si avanzamos en la dirección del objetivo por el camino A , C y B .

Observamos que para el nodo A la función h no es menor que la suma del costo a C , que es 1, con la heurística del nodo C que es 2.

Esto significa que el valor de $f(A) = g(A) + 4$ es mayor que el valor $f(C) = g(A) + 1 + 2$. Esto al avanzar en la profundidad observamos que el valor f no se incrementa, sino que disminuye.

Esto no es bueno para A^* , pues una heurística no monotonía puede llevarnos a soluciones subóptimas.

Admisibilidad de una heurística está implicada si la heurística es consistente.

Algoritmo A^*



└ Algoritmo A^*



Explicamos el algoritmo A^* en el pizarrón.

Algoritmo A^*

1. Memoria.
 - $O(b^d)$
 2. Tiempo.
 - $O(b^d)$
 3. Calidad.
 - Solución óptima.
 4. Completez.
 - Completo.
- El desempeño del algoritmo depende fuertemente de la calidad de la heurística.
 - En la práctica se caracteriza con un factor de ramificación b^* efectivo medido en forma empírica.
 - La optimalidad depende de la heurística.

2018-10-02

└ Algoritmo A^*

Algoritmo A

- | | |
|-----------------|--|
| 1. Memoria. | <ul style="list-style-type: none"> • $O(n^2)$ |
| 2. Tiempo. | <ul style="list-style-type: none"> • El desempeño del algoritmo depende fuertemente de la calidad de la heurística. |
| 3. Calidad. | <ul style="list-style-type: none"> • $O(n^2)$ • En la práctica se caracteriza con un factor de ramificación b^* efectivo medido en forma empírica. |
| 4. Complejidad. | <ul style="list-style-type: none"> • La optimalidad depende de la heurística. • Complejidad. |

Respecto de la complejidad computacional de A^* ,
el consumo de memoria es exponencial en la profundidad de la meta.
El tiempo también.

Al usar la cola de prioridad adolece de la complejidad adicional de la eliminación del mínimo en esta estructura de datos.

La calidad de la solución depende de que la heurística satisfaga dos condiciones:

la primera se denomina admisibilidad, la segunda monotonicidad.

Más adelante explicaremos que significan estas condiciones.

El algoritmo es completo, asumiendo que el estado meta es alcanzable.