

3.1 Introducción

La **heurística** es un método de valorar los nodos en el espacio de estados.

A mejor heurística, mayor coste

Decidir por que nodo de **ABIERTA** se continua es función del **control**.

Sin coste de control, hay que recorrer todo y el coste es alto.

A medida que aumenta el control, se reduce el de coste total de la búsqueda.

Si el control es demasiado costoso, comienza a aumentar el coste total.

3.2 Método búsquedas heurísticas

Búsqueda Primero el mejor: Procedimiento general de búsqueda en grafos, en cada paso se ordena **ABIERTA**, dando un valor según heurística (Contando o no el coste del camino parcial hasta el momento). Normalmente no considera ese coste.

Algoritmo A*: "Coste del camino más corto = ccc"

Definiciones: Dado un nodo n cualquiera del espacio de búsqueda, se definen:

$g^*(n)$: ccc del nodo inicial a n .

$h^*(n)$: ccc de n a un estado objetivo.

$f^*(n)$: ccc del nodo inicial a estado objetivo pasando por n : $f^*(n) = g^*(n) + h^*(n)$

$g(n)$: ccc **encontrado** del nodo inicial a n .

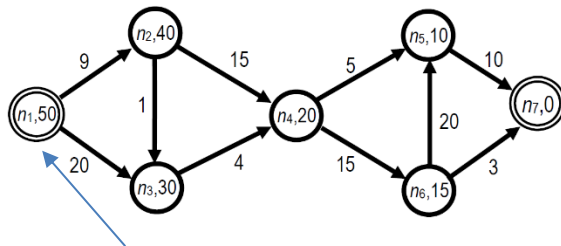
$h(n)$: **estimación** de $h^*(n)$, tal que $h(n) = 0$ si n fuese un estado objetivo.

La **función** que ordena **ABIERTA**: $f(n) = g(n) + h(n)$.

El valor de $f(n)$ cambia durante la ejecución, ($h(n)$ es fijo, $g(n)$ varía al expandir nuevos).

A^* es una variante del algoritmo de búsqueda general en grafos, donde la lista **ABIERTA** se ordena según $f(n)$ y las reorientaciones de **TABLA_A** se producen por cambios en $g(n)$.

Por ejemplo:



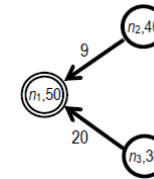
Nos indican cuál es el nodo inicial. En este caso n_1 .

Dentro de cada nodo se indica el valor de la estimación $h(n)$.

0º Se mete el nodo inicial n_1 en la lista **ABIERTA** y en **TABLA_A**:

$ABIERTA = \{n_1(0+50)\}$

1º Se expande el nodo n_1 de **ABIERTA**, tras ello, tenemos la siguiente situación:

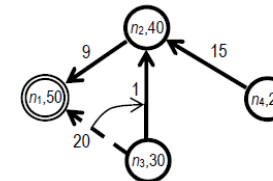


$ABIERTA = \{n_2(9+40), n_3(20+30)\}$

2º Se expande n_2 al ser el nodo de **ABIERTA** con menor valor de función de evaluación heurística: $f = g + h \rightarrow f = 9 + 40$.

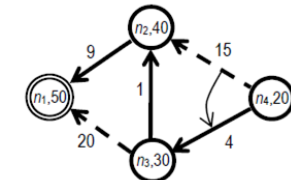
Se reorienta el mejor padre de n_3 , ya que antes era $n_1(20)$ y ahora es $n_2(1 + 9)$,

Entonces, siendo **TABLA_A** el grafo actualizado: **(Si se expandiese n_3 , el arco con valor 1 no se tendría en cuenta al no salir de n_3)**



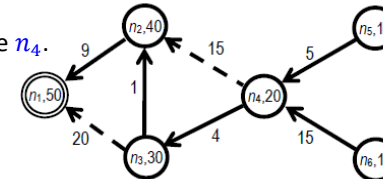
$ABIERTA = \{n_3(10+30), n_4(24+20)\}$

3º Se expande n_3 . Hay una orientación del mejor padre de n_4 , antes era $n_2 = 24 + 20$, y ahora pasa a ser, el nuevo coste de n_4 al inicial $g(n_4) = 4 + 1 + 9 = 14$, que se calcula a partir de los costes de los mejores arcos ascendentes hasta el momento:

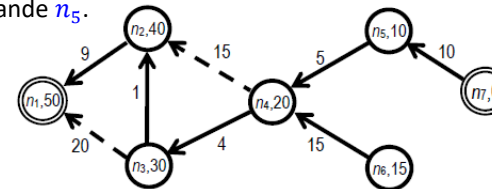


$ABIERTA = \{n_4(14+20)\}$

4º Se expande n_4 .



5º Se expande n_5 .



6º Se intenta expandir n_7 y se alcanza la meta, por lo que el algoritmo acaba.

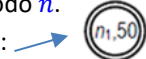
El camino solución es: $n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_4 \rightarrow n_5 \rightarrow n_7$ y su coste: $9 + 1 + 4 + 5 + 10 = 29$.

Fundamentos de inteligencia artificial

Teoremas:

- A^* es completo en grafo finitos

Definición: Una f. heurística h es admisible si $h(n) \leq h^*(n)$ para todo n .

Por ejemplo: En el grafo anterior, el coste de la estimación $h(n)$: 

Es mayor que el coste del mejor camino $h^*(n)$: 29. La función del ejemplo no es admisible.

(Se puede llegar a la solución óptima, pero no siempre se hará).

- Si la heurística es admisible, entonces A^* es admisible.

Definición: Una f. heurística h_1 está **más informada** que h_2 si:

Ambas son admisibles $\forall n, h_1(n) \geq h_2(n)$.

(Tiene más información). Mientras más suba, más se aproxima a los valores reales, esto provoca que se tengan que reorientar menos enlaces).

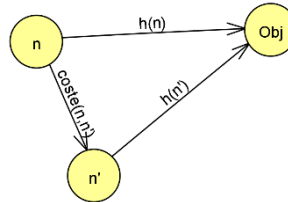
Definición: Una función heurística h es monótona si se cumple para todo par de nodos n y n' .

$$h(n) \leq \text{coste}(n, n') + h(n')$$

Por ejemplo: Esto quiere decir, que es monótona si se da el siguiente caso:

Si el valor de $h(n)$ es menor al coste de ir de n a n' más $h(n')$.

Si es monótona, no hay que reorientar enlace.



- Si se una heurística monótona, en A^* no es necesario reorientar enlaces

Variantes de A^* : **Relajación de optimalidad** a cambio de reducir coste.

- **Ponderación dinámica:** $f(n) = (1 - w) \cdot g(n) + w \cdot h(n)$, con $0 \leq w \leq 1$

$g(n)$ ccc (del nodo inicial a n), si solo es $g(n) \rightarrow$, búsqueda de coste uniforme.

$h(n)$ estimación del ccc del nodo n a un objetivo, si solo $h(n) \rightarrow$ primero el mejor.

A^* suma ambos conceptos: $f(n) = g(n) + h(n)$

La ponderación dinámica, se pondera con un valor entre 0 y 1. Se pondera con valor w la $h(n)$ y con $1 - w$ la $g(n)$. En función de w se parecerá más o menos a otras búsquedas:

Valor de w	Tipo de búsqueda
0	Coste uniforme
1/2	A^* estándar
1	Primero el mejor

w cambia según lo cerca que estemos de la solución (empieza con valor alto y va bajando).

Por ejemplo, se puede empezar con un valor alto, por el que tienden los nodos a acercarse al objetivo, y cuando estamos cerca del objetivo, se baja el valor, para afinar.

- **Algoritmo $A\epsilon^*$:** Tiene dos heurísticas, una de “grano grueso” y otra de “grano fino”.

Crea sublista **FOCAL** de **ABIERTA**, con nodos $f \leq (1 + \epsilon) \cdot \min$; \min : mínimo f en **ABIERTA**.

Es decir, se toman los mejores nodos, aquellos cuyo valor no supera ese mínimo. $(1 + \epsilon) \cdot \min$ guarda p.e. un 30% más.

Sobre **FOCAL**, se aplica la 2ª f. heurística. Todos sus nodos andan cerca de ser prometedores.

La segunda es más costosa pero solo se aplica a unos cuantos.

Variantes de A^* : **Búsqueda con memoria limitada** el tamaño de **ABIERTA**, se limita

- **Algoritmo IDA^* :** A^* crece de forma exponencial con el tamaño del problema. IDA^* amplía la búsqueda iterativa en profundidad asociando a cada nodo n , su valor de la función heurística propia del algoritmo A^* , $f(n) = g(n) + h(n)$.

IDA^* hace búsquedas en profundidad iterativamente desde el nodo raíz, aumentando en cada iteración la profundidad límite de las búsquedas.

La primera, a través de nodos n tales que $f(n) < f(\text{nodo raíz})$; si f fuese admisible, entonces $f(\text{nodo raíz})$ sería menor que el coste de la solución óptima.

Si un nodo no cumple esta condición, se descarta.

El menor f de descartados en la iteración actual, es el límite de profundidad de la siguiente.

Se guarda el menor de los nodos descartados. Y en vez de ser el límite $f(\text{nodo raíz})$, lo será $f(n \text{ de menor valor})$, que es mayor que el de la raíz (así se va aumentando solamente un poquito hasta encontrar la meta).

Normalmente, los hijos de cada nodo expandido, en orden en **ABIERTA** por su valor f .

Si h es admisible, IDA^* también lo es. El consumo de memoria pasa de ser exponencial (A^*) a proporcional al producto de la profundidad de la solución.

- **Algoritmo SMA^* :** Se acota la longitud de **TABLA_A**, y si cuando hay que expandir un nodo no hay espacio en **TABLA_A**, se descarta el menos prometedor de **ABIERTA**. (Recuerda en cada nodo el mejor de sus hijos que han sido descartados).

Algoritmos voraces: Toman decisiones irrevocables al explorar (sin alternativas). De los hijos del nodo actual, mediante heurístico, toma el **más prometedor**. Eficientes pero no admisibles.

Algoritmos de ramificación y poda: Interpretan cada nodo cómo un subconjunto de soluciones de todo el conjunto posibles de soluciones del problema original.

El nodo raíz representa todas las soluciones al problema original, y un nodo hoja será aquel que no puede ser expandido por contener una única solución, con un coste conocido y concreto.

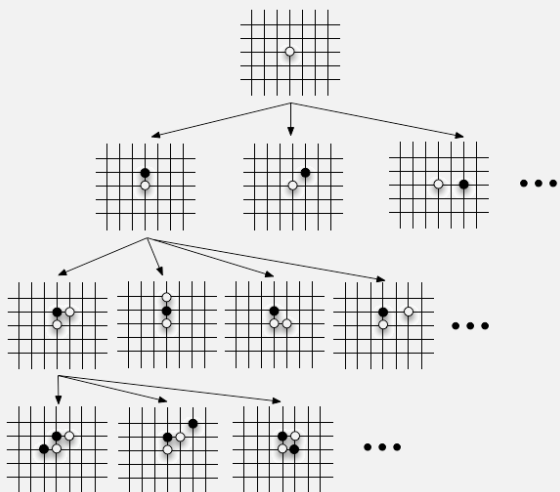
Fundamentos de inteligencia artificial

Y cada nodo no terminal del árbol tiene asociado una cota inferior del coste de la mejor solución (considerada cómo la de menor coste) contenida en el nodo.

Si la cota inferior de un nodo rebasa el menor de los costes de los nodos hoja encontrados hasta el momento en el resto del árbol, dicho nodo es podado.

Por ejemplo: Juegos de tablero:

- Raíz: Posición que se quiere analizar
- Hijos de un nodo: uno por cada posible movimiento
- Hojas: posiciones finales, (ganadoras, perdedoras o de otro tipo (no permitidas, tablas, etc))



En vez de genera el árbol completo (muy costoso), se va eligiendo el siguiente hijo a expandir (**ramificación**) siguiendo cierto criterio (menor coste esperado, por ejemplo) y evitando generar (**poda**) aquellos nodos que:

O bien se sepa que ya no pueden dar lugar a una solución

O bien se puede asegurar que, aún llevando a solución, es peor que otra ya conocida.

Suele ser en profundidad (pila = ABIERTA).

La eficiencia dependerá de lo ajustadas que sean las cotas inferiores obtenidas de forma heurística.

Son admisibles si recorren todo el espacio de búsqueda (excepto las partes podadas), hasta que **ABIERTA** se agote, si no, la condición de terminación es que el algoritmo pare después de expandir cierto número de nodos, se pierde admisibilidad y únicamente se puede devolver la mejor solución encontrada hasta el momento.

En resumen, se va desarrollando el grafo según la heurística que marca las cotas superior e inferior, y solo se va expandiendo esa parte.

Algoritmos de búsqueda local: Cuando el camino desde el nodo inicial al meta es irrelevante.

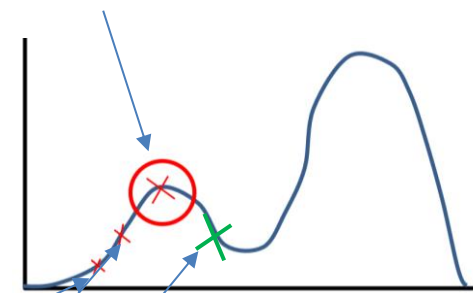
A partir de una **solución candidata** (nodo inicial) e ir haciendo **pequeñas modificaciones** (sucesores) hasta encontrar la solución óptima.

En problemas donde el meta tiene toda la info de solución del problema, sin necesitar el camino.

Consiste en hacer cambios en el estado actual que nos lleva a un estado vecino. Se calculan estos estados vecinos mediante una **regla de vecindad**. Los vecinos del actual se evalúan y se selecciona el mejor, que cumpla un criterio de aceptación.

El proceso se repite hasta encontrar un nodo que cumpla cierto **criterio de aceptación**.

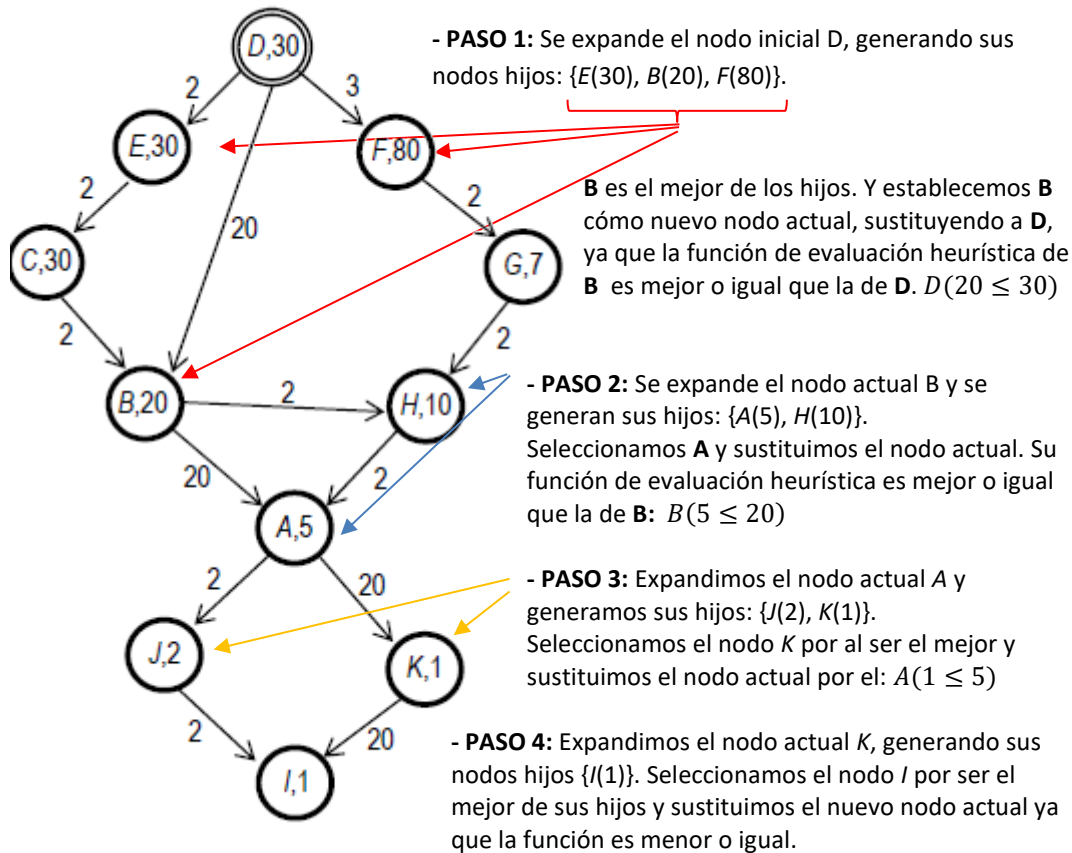
La función objetivo puede arrojar varios óptimos locales, lo que dificultará encontrar el óptimo global. Se pueden estancar en **máximos locales**.



Es decir, se va aumentando el valor de los nodos sucesores que va encontrando. Y Cuando llega al que está encerrado en un círculo, no continúa avanzando porque el siguiente nodo sería menor (en la gráfica en **verde**), perdiéndose tras varios nodos la opción de tomar el máximo de verdad.

Aunque proporciona soluciones aceptables en un tiempo bajo.

Búsqueda del gradiente: Desde cualquier nodo en el que estemos, nos vamos al mejor de los nodos sucesores, solamente si es mejor que el actual.



La búsqueda terminaría en este punto. El algoritmo devolvería el nodo I(1) como mejor nodo encontrado.

SE PODRIA DAR EL CASO DE NO TENER HIJOS MEJORES:

- **PASO 5:** Expandimos el nodo actual J y generamos sus hijos {I(3)}. Seleccionamos el nodo I ya que es el mejor de sus hijos (y el único por otra parte). A continuación, rechazamos el nodo I como nuevo nodo actual, ya que el valor de la función de evaluación heurística no es menor o igual: $J(3 \geq 2)$

La búsqueda terminaría en este punto. El algoritmo devolvería el nodo J(2) como mejor nodo encontrado.

Algoritmo de Temple simulado:

- Selecciona un nodo aleatorio de los vecinos del nodo actual
- Se comprueba si el nodo aleatorio, según un criterio de aceptación, se acepta o no. Es permite admitir con cierta probabilidad, alguna transición que empeore el valor de la función objetivo.

Esta probabilidad depende de dos parámetros.

La temperatura T y el incremento de energía $\Delta E = \text{coste}(\text{vecino seleccionado}) - \text{coste}(\text{estado actual})$. Esto puede evitar caer en óptimos locales (aunque no siempre).

- Al iniciar la búsqueda, la temperatura tiene valor alto, entonces, la probabilidad de aceptar un estado peor es grande. Y va decreciendo durante el proceso de búsqueda siguiendo un plan de enfriamiento, que puede ser más rápido o lento, lo que hace que a medida que avanza la búsqueda es más probable que solo se usen soluciones que mejoren o igualen la actual.

- Normalmente la probabilidad de obtener una buena solución es mayor si la temperatura inicial es alta y el plan de enfriamiento lento. Pero la búsqueda consume demasiado tiempo. Cómo inconveniente, tiene que hay que ajustar bien el valor de sus parámetros, y estos dependen del tipo de función objetivo y la distribución de sus óptimos locales.

Búsqueda Tabu:

Es un mecanismo de **memoria** que guarda ciertos nodos para evitar que se generen de nuevo (lista de movimientos tabú).

Usa un **criterio de aspiración**: Que son excepciones a la lista tabú, ya que, si un movimiento mejora la solución actual, se aplica cómo si no estuviera en la lista tabú.