BUSQUEDA CIEGA

BÚSQUEDA EN EL ESPACIO DE ESTADOS

Solucionar un problema mediante búsqueda: FORMULACIÓN + BÚSQUEDA + EJECUCIÓN

FORMULACIÓN DEL PROBLEMA

1. Definir ESTADOS
2. Especificar el ESTADO INICIAL
3. Especificar las ACCIONES que puede realizar el agente
   - REGLAS para acciones permitidas.
   - FUNCIÓN SUCESOR: estado actual → lista de estados directamente accesibles
4. Definir los estados OBJETIVO.
   - Definición extensiva: Lista
   - Definición intensiva: TEST DE OBJETIVO
5. Definir UTILIDAD: función del coste del camino.

CAMINO: Secuencia de estados conectados por acciones

ESPACIO DE ESTADOS: Conjunto de estados accesibles desde el estado inicial

Representación mediante un grafo conexo cuyos $\left\{ \begin{array}{l} \text{nodos} \equiv \text{estados} \\ \text{arcos} \equiv \text{acciones} \end{array} \right.$

SOLUCIÓN: Camino desde el estado inicial al estado(s) objetivo(s)

SOLUCIÓN ÓPTIMA: Solución con el mínimo coste.

ÁRBOL DE BÚSQUEDA

Casi siempre, la búsqueda es en un grafo de búsqueda (puede haber varios caminos desde el nodo inicial a un nodo dado).

Centrémonos en búsqueda en árbol (un solo camino desde el nodo raíz a un nodo dado).

- Los nodos de un AdB corresponden a estados de búsqueda.
- El nodo raíz de un AdB corresponde al estado de búsqueda inicial.
- Acciones: expandir el nodo de búsqueda actual: generar nodos hijo (nuevos estados) aplicando la función sucesor al nodo actual.
- Estado objetivo: un nodo correspondiente a un estado que satisface el test de objetivo.
- Utilidad: coste del camino desde el nodo raíz al nodo actual.

TERMINOLOGÍA

- **Nodo padre:** Nodo del árbol desde el cual se ha generado el nodo actual aplicando una sola vez la función sucesor.
- **Nodo ancestro:** Un nodo en el árbol desde el cual el nodo actual ha sido generado aplicando una o varias veces la función suceso
- **Profundidad:** Longitud del camino desde la raíz al nodo actual.
- **Nodo hoja:** Nodo generado que no se ha expandido aún.
- **Frontera:** Conjunto formado por nodos hoja.

PSEUDOCÓDIGO BÚSQUEDA EN ÁRBOL

```
problema = {nodo-raíz, expandir, test-objetivo}
estrategia

function búsqueda_en_árbol(problema, estrategia)
    Inicializar árbol-de-búsqueda con nodo-raíz
    Inicializar lista-abierta con nodo-raíz
    Iterar:
        If (lista-abierta está vacía):
            return fallo
        Else:
            Elegir de lista-abierta, de acuerdo estrategia, nodo a expar
            If(nodo satisface test-objetivo):
                return solución (camino nodo-raíz ~> nodo-actual)
            Else:
                Eliminar nodo de lista-abierta
                Expandir nodo
                Añadir nodos hijos a lista-abierta
```

ESTRATEGIAS DE BÚSQUEDA

- **BÚSQUEDA NO INFORMADA (CIEGA):**

Se usa solamente en la búsqueda la definición del problema.
Las diferentes estrategias de búsqueda difieren en el orden en el que los nodos se van expandiendo.

- Búsqueda primero en anchura
- Búsqueda de coste uniforme
- Búsqueda primero-en-profundidad
- Búsqueda de profundidad limitada
- Búsqueda primero en profundidad con profundidad iterativa
- Búsqueda bidireccional.

- **BÚSQUEDA INFORMADA (HEURÍSTICA)**

Usa heurística (estimaciones de cómo de lejos está un estado dado del estado objetivo) para guiar la búsqueda.

RENDIMIENTO

Rendimiento resolviendo problemas →
- completitud
- optimalidad
- costes
  - → coste del camino: función utilidad
  - → coste de la búsqueda
    - → complejidad temporal
    - → complejidad espacial (uso memoria)

Coste de la búsqueda (análisis complejidad)
- Factor de ramificación (b): nº máximo de sucesores de cualquier nodo.
- Profundidad del nodo objetivo más superficial (d)
- Profundidad máxima del árbol de búsqueda (m)
- Coste mínimo de una acción (ε)

Hipótesis:
- Factor de ramificación (b) finito.
- Desde el punto de vista del análisis del coste, todas las operaciones tienen el mismo coste.
- La profundidad máxima del árbol de búsqueda (m) puede ser infinita.
- Coste del camino = suma de costes de cada paso (que no son negativos).

▸ BÚSQUEDA PRIMERO EN ANCHURA

Expandir todos los nodos de una profundidad dada antes de expandir los nodos de una profundidad mayor.

- Se puede implementar utilizando una <u>cola FIFO</u> para lista-abierta.
- En cuanto al rendimiento, el algoritmo es <u>completo</u>. Es óptimo solo si el coste es función de la profundidad y no decreciente.

Complejidad temporal y espacial:

↳ nodos generados $= b + b^2 + \dots + (b^{d+1} - b) = \dfrac{b^{d+2} - b^2}{b-1} = O(b^{d+1})$

↳ nodos expandidos $= 1 + b + \dots + b^d - 1 = \dfrac{b^{d+1} - b}{b-1} = O(b^d)$

BÚSQUEDA DE COSTE UNIFORME

Expandir el nodo con el coste de camino más bajo.

- Se puede implementar ordenando lista-abierta de acuerdo a sus costes de camino. Se expande primero el nodo de menor coste.
- Rendimiento:
  Es completo y óptimo si y solo si el coste de cada paso es mayor que cero (e.d. coste crece si long. camino crece). Si hubiese un bucle con coste cero podría entrar en un bucle infinito.
- Complejidad:
  Caso peor $\sim O(b^{\lceil C^*/\varepsilon \rceil})$ con $C^* \equiv$ coste del camino de la sol. óptima, $\varepsilon \equiv$ coste mínimo (>0) de una acción.

- Si todos los pasos tienen igual coste, la búsqueda de coste uniforme es equivalente a búsqueda primero en anchura.

BÚSQUEDA PRIMERO EN PROFUNDIDAD

Expandir primero los nodos más profundos de la frontera.

Implementación → cola LIFO
→ recursión por cada uno de los hijos del nodo expandido.

Rendimiento: no es completa (profundidad infinita en algunos casos) y no es óptima.

Complejidad: asumamos m profundidad máxima.
→ temporal ⇒ peor caso $= O(b^m)$
→ espacial ⇒ se debe llevar cuenta del camino desde raíz hasta hoja + hermanos no expandidos $O(bm+1)$

▶ BÚSQUEDA CON VUELTA ATRÁS (BACKTRACKING)

Variante de búsqueda con profundidad limitada, con uso eficiente de la memori

• Solo se genera un sucesor en cada expansión y en cada nodo parcialmente expandido se recuerda cuál es el siguiente sucesor a generar.
$\Longrightarrow O(m)$ estados

• Generar sucesor modificando el estado actual (en vez de copiar + modificar
Se deben poder deshacer modificaciones cuando se vaya hacia atrás para generar los siguientes sucesores.
$\Longrightarrow$ un sólo estado + $O(m)$ acciones.

▶ BÚSQUEDA DE PROFUNDIDAD LIMITADA

Expandir primero los nodos más profundos de la frontera, hasta una profundidad máxima (ℓ).

• Posible implementación: igual que busq-prim-prof., asumiendo que los nodo de profundidad igual a ℓ no tienen sucesores.

• Rendimiento: no es completo si $\ell < d$
no es óptimo si $\ell > d$
es óptimo si $\ell = d$ y los costes de cada paso son iguales.

• Complejidad → temporal: peor caso = $O(b^\ell)$
→ espacial: $O(b\ell)$

▶ BÚSQUEDA PRIMERO-EN-PROFUNDIDAD CON PROFUNDIDAD ITERATIVA

Expandir nodos hasta una profundidad máxima, e ir incrementando esta profundidad límite.

• Implementación: combinar con búsqueda de prof. limitada con $\ell = 0, 1, 2 \ldots$

• Rendimiento: completo si el factor de ramificación (b) es finito.
óptimo si los costes de cada paso son iguales.

• Complejidad → temporal: $(d)b + (d-1)b^2 + (d-2)b^3 + \cdots + 2b^{d-1} + b^d$
puede ser menor que búsqueda en anchura
→ espacial: $O(bd)$

⊛ Es la estrategia de búsqueda ciega preferible cuando el espacio de búsqueda es grande y se desconoce el valor d.

⊛ Búsqueda con alargamiento iterativo: limitar el coste máximo del camino e irlo incrementando, en vez de incrementar el límite a la profundidad (normalmente costoso).

▶ BÚSQUEDA BIDIRECCIONAL

Combinar búsqueda hacia delante (del estado inicial al final) y búsqueda atrás (del estado final al estado inicial).

• Implementación: se encuentra la solución cuando el nodo a expandir en una búsqueda está en el conjunto frontera del otro árbol. Puede usarse en combinación con cualquier otra estrategia de búsqueda.

• Rendimiento → completo si b finito
→ óptimo si los costes de cada paso son iguales.

• Complejidad → temporal: $O(b^{d/2})$
→ espacial: se puede mantener en memoria al menos un árbol de búsqueda $O(b^{d/2})$.

Dificultades:
- La función predecesor debe ser eficiente
- Definición implícita de estados objetivo

▶ RESUMEN

Criterio	PRIMERO ANCHURA	COSTE UNIFORME	PRIMERO PROFUND.	PROFUND. LIMITADA	PROFUND. ITERATIVA
¿completo?	Sí	Sí	No	Sí ($l \geq d$)	Sí
Tiempo	b^{d+1}	$b^{\lceil C^*/\varepsilon \rceil}$	b^m	b^l	b^d
Espacio	b^{d+1}	$b^{\lceil C^*/\varepsilon \rceil}$	bm	bl	bd
¿óptimo?	Sí	Sí	No	No	Sí

Observación

→ BÚSQUEDA EN ÁRBOL: Sin eliminación estados repetidos

→ BÚSQUEDA EN GRAFO: Con eliminación estados repetidos

BÚSQUEDA INFORMADA

Las estrategias de búsqueda ciega (no informada) son generalmente muy ineficientes. El uso de conocimiento específico sobre el problema para guiar la búsqueda puede mejorar enormemente la eficiencia.

Versión informada de algoritmos de búsqueda general: BÚSQUEDA 1º EL MEJ

- Búsqueda avariciosa
- Búsqueda A*
- Búsqueda heurística con memoria acotada:
  - IDA* (A* con profundidad iterativa)
  - RBFS (búsqueda 1º el mejor recursiva)
  - MA* (A* con memoria acotada)
  - SMA* (MA* simplificada)

BÚSQUEDA PRIMERO EL MEJOR

Elegimos de lista-abierta el nodo que expandiremos de acuerdo a una función de evaluación [f(n)] que da el coste del camino <u>menos costos</u> que va desde el nodo n al objetivo.

- Implementación: usar búsqueda-en-grafo con una cola de prioridad para lista-abierta. Los nuevos nodos se insertan en la cola en orden ascendente del coste.
- Rendimiento: por definición es óptima, completa y tiene la menor complejidad posible, pero no es una búsqueda.

<u>DEFINICIÓN</u>: Decimos que la función heurística h(n) es ADMISIBLE si nunca sobreestima el coste de alcanzar el objetivo, e.d.:

$h(n) \leq h^*(n)$ $\forall n$ nodo donde $h^*(n) \equiv$ coste real camino óptimo estado actual $\longrightarrow$ meta

TEOREMA: Si se usa búsqueda en árbol (sin eliminación de estados repetidos) y h es admisible, entonces A^* es COMPLETA y ÓPTIMA.

demostración

Sea C^* coste solución óptima

Consideramos n_2 un nodo objetivo no-óptimo (e.d. $g(n_2) > C^*$, $h(n_2)=0$) que está en la frontera.

$f(n_2) = g(n_2) + h(n_2) = g(n_2) > C^* \Rightarrow f(n_2) > C^*$ [1]

Consideremos el nodo n de la frontera que está en un camino de una
de una solución óptima. Dado que n está en el camino de una
solución óptima $g(n) = g^*(n)$, y como h es admisible $h(n) \leq h^*(n)$

$f(n) = g(n) + h(n) \leq g^*(n) + h^*(n) = C^* \Rightarrow f(n) \leq C^*$ [2]

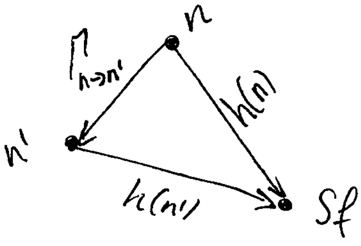
[1] + [2] $\Rightarrow f(n) \leq C^* < f(n_2)$ y se explora n antes que n_2.

EFINICIÓN: Una función heurística $h(n)$ es MONÓTONA si se satisface la esigualdad triangular: $h(n) \leq coste(n \to n') + h(n')$ $\forall n, n'$ n' sucesor de n.

equivalente: $h(n) \leq \Gamma_{n \to n'} + h(n')$

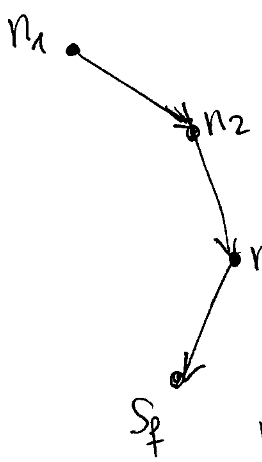
EOREMA: h monótona $\Rightarrow$ h admisible

demostración



$h(n) \leq \Gamma_{n \to n'} + h(n')$

definición de monotonía



$\left.\begin{array}{l} h(n_1) \leq \Gamma_{n_1 \to n_2} + h(n_2) \\ h(n_2) \leq \Gamma_{n_2 \to n_3} + h(n_3) \end{array}\right\} h(n_1) \leq \Gamma_{n_1 \to n_2} + \Gamma_{n_2 \to n_3} + h(n_3)$

$h(n_3) \leq \Gamma_{n_3 \to S_f} + \cancel{h(S_f)}^{0}$

$\Rightarrow h(n_1) \leq \Gamma_{n_1 \to n_2} + \Gamma_{n_2 \to n_3} + \Gamma_{n_3 \to S_f}$

$h(n_1) \leq$ suma costes en el camino que va de n a S_f (sea óptimo o no) $\xrightarrow{(*)} h(n_1) \leq h^*(n_1)$

(*) entre esos caminos está el óptimo

TEOREMA: h monótona $\Rightarrow$ $f(n)$ a lo largo del camino buscado por A^* son no decrecientes.

demostración

Supongamos que n' es sucesor de n

$f(n') = g(n') + h(n') = g(n) + coste(n \to n') + h(n') \geq g(n) + h(n) = f(n)$

$\Rightarrow f(n') \geq f(n)$ ∎

TEOREMA: A^* usando búsqueda en grafo (eliminación estados repetidos) con heurística monótona es completa y óptima.

demostración

Dado que $f(n)$ es no-decreciente el primer nodo objetivo expandido debe ser el correspondiente a la solución óptima.

COMPLEJIDAD DE A^*

Complejidad temporal: exponencial para h arbitrario $O(b^{\tilde{d}})$ con $\tilde{d} = \frac{C^*}{\varepsilon}$ donde $C^* \equiv$ coste óptimo y $\varepsilon =$ mínimo coste por acción.

Subexponencial si $|h(n) - h^*(n)| \leq O(\log h^*(n))$

Complejidad espacial: igual a la complejidad temporal (se mantienen los nodos en memoria). Normalmente es el factor limitante.

Debido al gran requerimiento de memoria, A^* no es un algoritmo práctico para problemas grandes.

BÚSQUEDA IDA*

Búsqueda A^* con profundidad iterativa

Realizar una búsqueda primero-en-profundidad con una profundidad límite de f, e ir aumentando este valor.

Propiedades:

- Si h es monótona $\Rightarrow$ IDA* es completa y óptima.
- Complejidad $\longrightarrow$ espacial: $O(bd)$; $d = C^*/\varepsilon$
  - $\longrightarrow$ temporal: En el peor caso, un sólo nodo es expandido en cada iteración. Asumiendo que el último nodo que se expande es el nodo solución, el número de iteraciones es $1+2+\cdots+N \sim O(N$

heurísticas admisibles

DEFINICIÓN: Se dice que h_1 DOMINA a h_2 si $\forall n: h_1(n) \geq h_2(n)$

Los algoritmos A^* y IDA* convergen antes cuanto mejor sea la heurística

BÚSQUEDA CON ADVERSARIOS

TIPOS DE JUEGOS: Criterios de clasificación

Por número de jugadores
- → dos jugadores
- → multijugador (estrategias mixtas)

Por función de utilidad
- → suma cero: suma de utilidades de los agentes es cero (ajedrez, damas)
- → suma constante: equivalente a los de suma cero con normalización
- → suma variable: no suma cero (monopoly)

Por la información que tienen los jugadores
- → información perfecta (ajedrez, damas)
- → información parcial (juegos de cartas)

Elementos al azar
- → deterministas (ajedrez, damas)
- → estocásticos

Tiempo limitado/ilimitado
Movimientos limitados/ilimitados

PROBLEMA DE BÚSQUEDA CON ADVERSARIOS

- Estado inicial
- Función sucesor: estado actual ⟶ estado sucesor
- Test terminal: función que determina si el estado del juego es terminal (e.d., si el juego ha finalizado)
- Utilidad (función objetivo o de pago): valoración numérica de los estados terminales.
- Árbol del juego: estado inicial + movimientos legales alternados

Consideramos un juego con dos jugadores: MAX y MIN. Max mueve primero y ambos van alternando sus turnos.

En el árbol
- nodos de profundidad par → MAX
- nodos de profundidad impar → MIN

Terminología: dupla de prof. $K \equiv$ nodos de profundidad $2K$ y $2K+1$.

- Estrategia óptima MAX: resultado al menos tan bueno como cualquier otra estrategia, asumiendo que MIN es un rival infalible.
- Estrategia minimax: usar el valor minimax de un nodo para guiar la búsqueda

ALGORITMO MINIMAX

- COMPLETO sólo si el árbol de juego es finito (puede haber estrategias óptimas para árboles infinitos).
- ÓPTIMO sólo si el oponente es óptimo
- COMPLEJIDAD TEMPORAL exponencial $O(b^m)$ $m=$ prof. máxima árbol
- COMPLEJIDAD ESPACIAL: lineal si se usa DFS (profundidad) $O(bm)$

Nodos MAX: maximizan Nodos MIN: minimizan

ALGORITMO PODA ALFA-BETA

Mejora del algoritmo MINIMAX:

α es el valor de la mejor alternativa para MAX encontrada hasta el momento (e.d., la de mayor valor)

β es el valor de la mejor alternativa para MIN encontrada hasta el momento (e.d., la de menor valor)

ALGORITMO DE PODA

en MAX: $\alpha \longleftarrow \max(\alpha, \beta\text{'s sucesores})$

en MIN: $\beta \longleftarrow \min(\beta, \alpha\text{'s sucesores})$

PODA CUANDO

$\alpha \geq \beta$

$[\alpha, \beta]$ intervalo de incertidumbre (contiene el valor MINIMAX)

$\alpha \leq$ VALOR MINIMAX $\leq \beta$

VALORES MINIMAX:

en MAX es α

en MIN es β

Observación: interesante algoritm EXCEPTI-MINIMAX

Minimax con prome probabilista.