

Resolución de problemas

Búsquedas en el espacio de estados

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

3 de julio de 2021



Búsqueda general

- 1 Búsqueda general
- 2 Búsquedas simples
- 3 Direcciones
- 4 Búsquedas iterativas

Temas

- 1 Búsqueda general
 - Resolución de problemas
 - Implementación

Resolución de problemas mediante búsquedas

- La IA utiliza técnicas de búsqueda en árboles o grafos para resolver problemas $\mathcal{P} = (\Sigma, s_i, g)$.
- La búsqueda se realiza sobre el sistema de transiciones de estados $\Sigma = (S, A, \gamma)$ visto como un grafo.
- Excepto para problemas triviales, el espacio de búsqueda no se genera completamente en memoria, pues éste **no cabe**.
- El nodo inicial es el correspondiente al estado s_i .
- El objetivo de la búsqueda es encontrar una ruta, descrita por una secuencia de acciones $\pi = \langle a_1, \dots, a_k \rangle$, desde s_i hasta algún con nodo con $s \models g$.

Temas

- 1 Búsqueda general
 - Resolución de problemas
 - Implementación

Nodo de una búsqueda

Un *Nodo* que contendrá los elementos siguientes (Wickler y Tate 2015):

- *Estado*. La representación de un estado en el espacio de estados.
- *Nodo padre*. Una referencia a su nodo predecesor inmediato en el árbol de búsqueda.
- *Acción*. La acción que, al realizarse en el estado del nodo padre produce el estado de este nodo.
- *Costo del camino*. El costo total del camino que lleva a este nodo.
- *Profundidad*. La profundidad de este nodo en el árbol de búsqueda.

Búsquedas simples

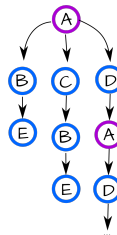
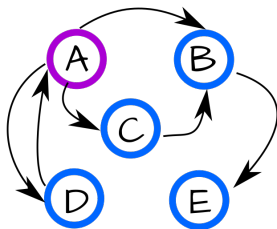
- 1 Búsqueda general
- 2 Búsquedas simples
- 3 Direcciones
- 4 Búsquedas iterativas

Temas

- 2 Búsquedas simples
 - Búsqueda en un árbol
 - Estrategias de búsqueda
 - Búsqueda en amplitud
 - Búsqueda en profundidad

Características

- Aunque el sistema de transiciones Σ contenga ciclos, no se hace ningún esfuerzo por detectarlos.
- Cada estado es generado cuando es alcanzable por un nodo en el grafo de búsqueda.
- Estados en ciclos pueden ser generados en varias ocasiones.



Pseudocódigo

Algoritmo 1 Búsqueda en un árbol.

```
1: function BÚSQUEDAENÁRBOL(problema, estrategia)
2:   margen  $\leftarrow$  { new NODOBÚSQUEDA(problema.si) }
3:   loop
4:     if VACÍA(margen) then return Fallo
5:     nodo  $\leftarrow$  seleccionaDe(margen, estrategia)
6:     if problema.PruebaEsMeta(nodo.estado) then
7:       return caminoA(nodo)
8:     margen  $\leftarrow$  margen + expande(problema, nodo)
```

Temas

- 2 Búsquedas simples
 - Búsqueda en un árbol
 - Estrategias de búsqueda
 - Búsqueda en amplitud
 - Búsqueda en profundidad

Varias estrategias

Para generar las diferentes estrategias de búsqueda se puede:

- Cambiar la estructura de datos adecuada para alterar el orden en que se selecciona el nodo siguiente,
- O en la línea 8, modificar la posición en la estructura donde son agregados los nodos a expandir.
 - *Búsqueda en profundidad*: la **lista abierta** es una **pila**.
 - *Búsqueda en amplitud*: la lista abierta es una **cola**.
- Agregar el uso de una **lista cerrada** para no volver a expandir nodos que lleven a estados ya visitados \Rightarrow *búsqueda en grafos*.
- Limitar temporalmente el alcance del recorrido:
 - *Profundidad iterativa*: Incrementar gradualmente la profundidad máxima que puede tener un nodo a generar (DFS).
 - *Amplitud iterativa*: Incrementar gradualmente el número de hijos (BFS).

Temas

2 Búsquedas simples

- Búsqueda en un árbol
- Estrategias de búsqueda
- Búsqueda en amplitud
- Búsqueda en profundidad

Búsqueda en amplitud

- Utiliza una **cola** como estructura auxiliar
- Los nodos hijos son generados y formados al final de la cola cada vez que se visita a un nodo.
- Los nodos son visitados por niveles.

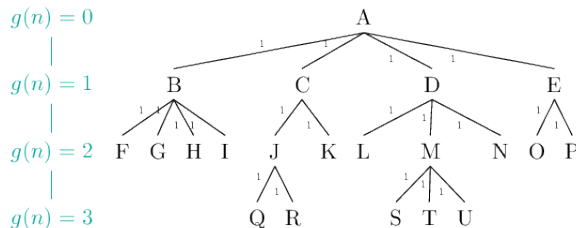


Figura: Orden en que son visitados los nodos: **A**, **B**, **C**, **D**, **E**, **F**, **G**, **H**, **I**, **J**, **K**, **L**, **M**, **N**, **O**, **P**, **Q**, **R**, **S**, **T**, **U**.

Temas

- 2 Búsquedas simples
 - Búsqueda en un árbol
 - Estrategias de búsqueda
 - Búsqueda en amplitud
 - Búsqueda en profundidad

Búsqueda en profundidad

- Utiliza una **pila** como estructura auxiliar
- Los nodos hijos son generados y formados en el tope de la pila cada vez que se visita a un nodo.
- Los nodos hasta el fondo del grafo y luego se pasa a otra rama.

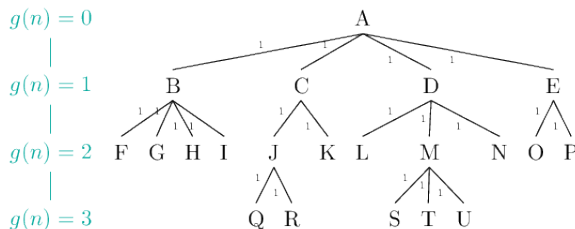


Figura: Orden en que son **visitados** los nodos: **A, B, F, G, H, I, C, J, Q, R** etc. Orden en que fueron **generados**: A, B, C, D, E, F, G, H, I, J, K, Q, R, L, M, N, S, T, U, O, P.

Direcciones

- 1 Búsqueda general
- 2 Búsquedas simples
- 3 Direcciones**
- 4 Búsquedas iterativas

Temas

3 Direcciones

- Búsqueda hacia adelante
- Búsqueda hacia atrás
- Búsqueda bidireccional

Búsqueda hacia adelante

Algoritmo 2 Búsqueda hacia adelante

```
1: function BÚSQUEDAHACIAADELANTE( $O, s_i, g$ )
2:   estado  $\leftarrow s_i$ 
3:   plan  $\leftarrow \langle \rangle$ 
4:   loop
5:     if estado.satisface( $g$ ) then
6:       return plan
7:     aplicables  $\leftarrow \{\text{instancias de } O \text{ aplicables en estado}\}$ 
8:     if aplicables.EsVacía() then return Fallo
9:     acción  $\leftarrow \text{aplicables.eligeUna}()$ 
10:    estado  $\leftarrow \gamma(\text{estado}, \text{acción})$ 
11:    plan  $\leftarrow \text{plan} \bullet \langle \text{acción} \rangle$ 
```

Temas

3 Direcciones

- Búsqueda hacia adelante
- Búsqueda hacia atrás
- Búsqueda bidireccional

Búsqueda hacia atrás

- Permite iniciar en algún nodo que satisfaga g .
- El algoritmo es análogo al anterior, pero se define:

Acción relevante Dada la función meta g , α es relevante si $g \cap \text{efectos}^+(\alpha) \neq \emptyset$ y $g \cap \text{efectos}^-(\alpha) = \emptyset$.

Estados de regresión

$$\Gamma^{-1}(g) = \{\gamma^{-1}(g, \alpha) | \alpha \in A \text{ y } \alpha \text{ es relevante para } g\}. \quad (1)$$

Estados precursores (El nombre precursores fue asignado para esta diapositiva)

$$\hat{\Gamma}^{-1}(g) = \Gamma^{-1}(g) \cup \Gamma^{-1}(g)^2 \cup \dots \quad (2)$$

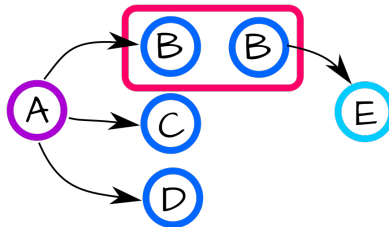
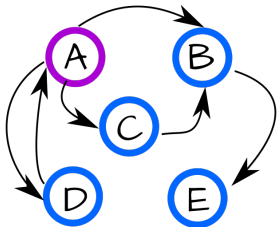
Temas

3 Direcciones

- Búsqueda hacia adelante
- Búsqueda hacia atrás
- Búsqueda bidireccional

Búsqueda bidireccional

- Se reemplaza la verificación de la función meta por una revisión para ver si las fronteras de las dos búsquedas se intersectan.



Búsquedas iterativas

- 1 Búsqueda general
- 2 Búsquedas simples
- 3 Direcciones
- 4 Búsquedas iterativas

Temas

- 4 Búsquedas iterativas
 - Profundidad iterativa
 - Amplitud iterativa

Profundidad limitada

Algoritmo 3 Profundidad limitada

```
1: function PLIMITADA( $\mathcal{P} = (\Sigma, s_i, g)$ ,  $p_{\max}$ , profundidad, plan)
2:   if  $g(s_i)$  then return plan
3:   if profundidad  $\geq p_{\max}$  then return fallo
4:   acciones  $\leftarrow$  aplicables( $s_i$ )
5:   if acciones =  $\emptyset$  then return fallo
6:   profundidad  $\leftarrow$  profundidad + 1
7:   for each acción in acciones do
8:     estado  $\leftarrow \Sigma.\gamma(\text{estado}, \text{acción})$ 
9:      $\mathcal{P}' = (\Sigma, \text{estado}, g)$ 
10:    r  $\leftarrow$  PLIMITADA( $\mathcal{P}'$ ,  $p_{\max}$ , profundidad, anexa(plan, acción))
11:    if  $r \neq$  fallo then return r
12:  return fallo
```

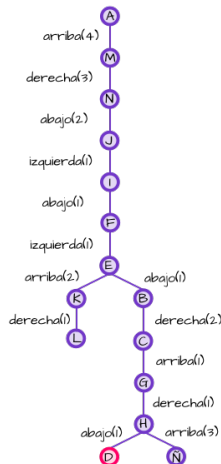
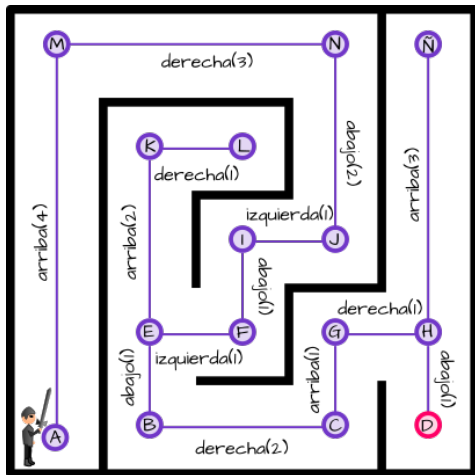
▷ Solución encontrada
▷ Profundidad límite
▷ Fin de la rama
▷ Desmontando llamadas
▷ No hay solución hasta profundidad

Profundidad iterativa

Algoritmo 4 Profundidad iterativa

```
1: function PROFUNDIDADITERATIVA( $\mathcal{P} = (\Sigma, s_i, g)$ )
2:   profundidad  $\leftarrow$  profundidad_inicial
3:   plan  $\leftarrow \emptyset$ 
4:   while plan =  $\emptyset$  do
5:     plan  $\leftarrow$  PLIMITADA( $\mathcal{P}$ , profundidad, 0, plan)
6:     if fallo then
7:       if profundidad = max_profundidad( $\Sigma$ ) then return fallo
8:       plan  $\leftarrow \emptyset$ 
9:       profundidad  $\leftarrow$  profundidad + incremento
10:    else
11:      return plan
```

Ejemplo



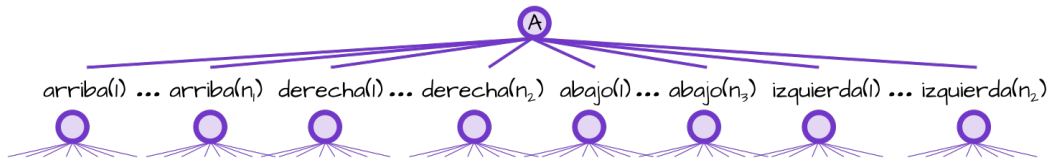
- p=0 A
- p=3 A, M, N, J
- p=6 A, M, N, J, I, F, E
- p=9 A, M, N, J, I, F, E, K, L, B, C, G
- p=12 A, M, N, J, I, F, E, K, L, B, C, G, H, D

Temas

- 4 Búsquedas iterativas
 - Profundidad iterativa
 - Amplitud iterativa

Amplitud iterativa

- Funciona en forma semejante a profundidad iterativa, pero lo que se limita es el **número de hijos**.
- Es útil cuando existe un gran número de acciones aplicables sobre cada estado (factor de ramificación alto).
- Por sí misma no resuelve el problema de las ramas con profundidad infinita, por lo que en la práctica se usaría *amplitud con profundidad iterativa*, es decir, se limita el número de hijos y la profundidad de la búsqueda.



Referencias I

-  Ghallab, Malik, Dana Nau y Paolo Traverso (2004). *Automated Planning, Theory and Practice*. Morgan Kaufmann Publishers.
-  Russell, Stuart y Peter Norving (2010). *Artificial Intelligence, A Modern Approach*. Ed. por Michael Hirsch. 2a. Pearson Prentice Hall.
-  Wickler, Gerhard y Austin Tate (ene. de 2015). *Artificial Intelligence Planning*. English. University of Edinburgh. URL: <https://www.coursera.org/course/aiplan>.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual

