Resolución de problemas

Búsquedas en el espacio de estados

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

3 de julio de 2021

Búsqueda general

Temas

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 correpondiente al estado s_i.
- El objetivo de la búsqueda es encontrar una ruta, descrita por una secuencia de acciones $\pi =< a_1, ..., a_k >$, desde s_i hasta algún con nodo con $s \vDash g$.

Temas

Nodo de una búsqueda

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

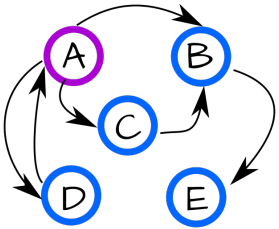
- *Estado.* La representación de un estado en el espacio de estados.
- *Nodo padre.* Una referencia a su nodo predecesor inmediato en el árbol de búsqueda.
- *Acción.* La acción que, al realizarse en el estado del nodo padre produce el estado de este nodo.
- *Costo del camino.* El costo total del camino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

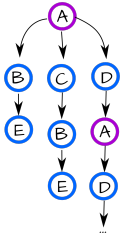
Búsquedas simples

Temas

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 ← { new NODOBÚSQUEDA(problema.s_i)}
3:     loop
4:         if VACÍA(margen) then return Fallo
5:         nodo ← seleccionaDe(margen, estrategia)
6:         if problema.PruebaEsMeta(nodo.estado) then
7:             return caminoA(nodo)
8:         margen ← margen + expande(problema, nodo)
```

Temas

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

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.

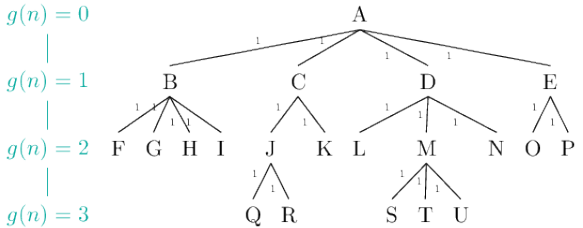


Figura: Orden en que son visitados los nodos: **A**,B,C,D,E, F,G,H,I|J,K|L,M,N|O,P, Q,R|S,T,U.

Temas

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.

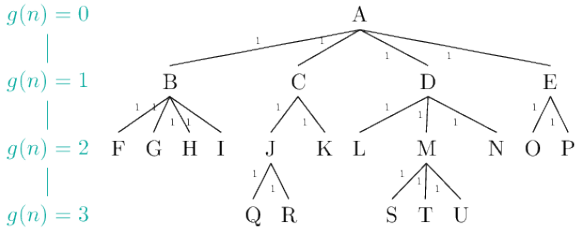


Figura: Orden en que son **visitados** los nodos: **A,B,F**,G, H, I, C,J,Q, R etc. Orden en que fueron **generados**: A, B, C, D, E, F, G, H, I, J, K, Q, R, L, M, N, S, T, U, O, P.

Direcciones

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 ← s_i
3:      plan ←<>
4:      loop
5:          if estado.satisface(g) then
6:              return plan
7:          aplicables ← {instancias de O aplicables en estado}
8:          if aplicables.EsVacía() then return Fallo
9:          acción ← aplicables.eligeUna()
10:         estado ← γ(estado, acción)
11:         plan ← plan • < acción >
```

Temas

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, a es relevante si $g \cap efectos^{+}(a) \neq \emptyset$ y $g \cap efectos^{-}(a) = \emptyset$.

Estados de regresión

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

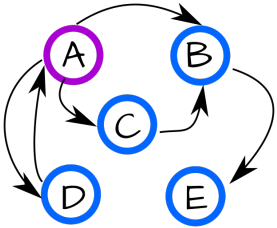
Estados precursores (El nombre precursores fue asignado para esta diapositiva)

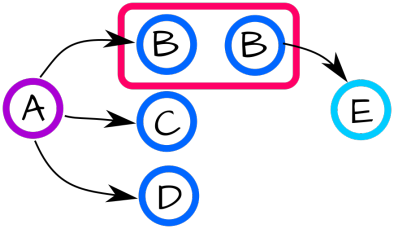
$$\hat{\Gamma}^{-1}(g) = \Gamma^{-1}(g) \cup \Gamma^{-1}(g)^2 \cup ... \quad (2)$$

Temas

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

Temas

Profundidad limitada

Algoritmo 3 Profundidad limitada

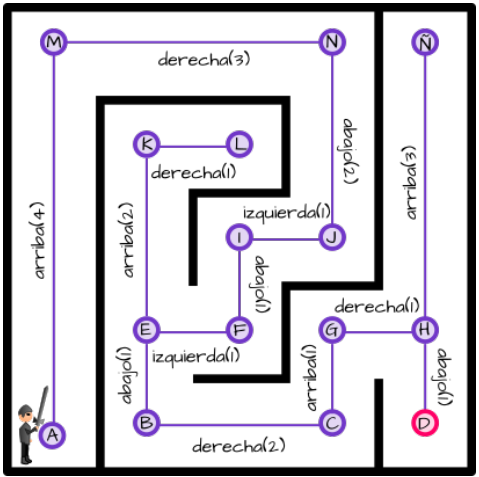
```
1:  function PLIMITADA(𝒫 = (Σ, s_i, g), p_max, profundidad, plan)
2:      if g(s_i) then return plan                          ▷ Solución encontrada
3:      if profundidad ⩾ p_max then return fallo            ▷ Profundidad límite
4:      acciones ← aplicables(s_i)
5:      if acciones = ∅ then return fallo                   ▷ Fin de la rama
6:      profundidad ← profundidad + 1
7:      for each acción in acciones do
8:          estado ← Σ.γ(estado, acción)
9:          𝒫′ = (Σ, estado, g)
10:         r ← PLIMITADA(𝒫′, p_max, profundidad, anexa(plan, acción))
11:         if r ≠ fallo then return r                      ▷ Desmontando llamadas
12:     return fallo                                        ▷ No hay solución hasta profundidad
```

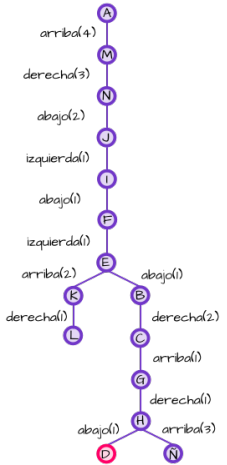
Profundidad iterativa

Algoritmo 4 Profundidad iterativa

```
1: function PROFUNDIDADITERATIVA(P = (Σ, s_i, g))
2:     profundidad ← profundidad_inicial
3:     plan ← ∅
4:     while plan = ∅ do
5:         plan ← PLIMITADA(P, profundidad, 0, plan)
6:         if fallo then
7:             if profundidad = max_profundidad(Σ) then return fallo
8:             plan ← ∅
9:             profundidad ← 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

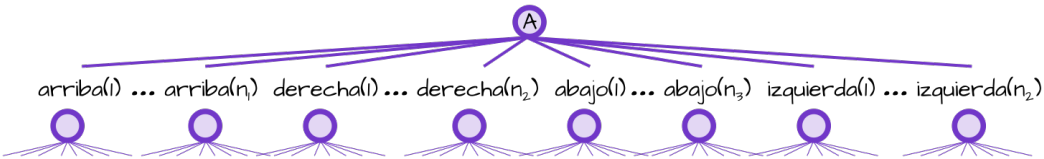
p=9 A, M, N, J, I, F, E, K, L, B, C, G

p=12 A, M, N, J, I, F, E, K, L, B, C, G, H, D

Temas

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