Búsqueda en IA

- Todo problema en el que es necesario **encontrar** una solución puede ser formulado como un problema de búsqueda.
- Un algoritmo se dice **de búsqueda** si se mueve a través de un espacio de búsqueda para encontrar la solución.
- Se usa un algoritmo de búsqueda en problemas en donde no se tiene una solución algorítmica.
- Posibles ejemplos: planificar un viaje, jugar ajedrez, resolver un puzzle, etc.

Mundos determinísticos con un solo agente

- Un espacio de estados $\mathcal{S}$.
- Un conjunto $\mathcal{A}$ de operaodres. Un operador $a\in\mathcal{A}$ es una función parcial:
  $$a:\mathcal{S}\rightarrow\mathcal{S}$$
- Por cada estado, un conjunto $A(s)\subseteq\mathcal{A}$ de operadores aplicables en s, Si $a\in\mathcal{A}$ entonces $a(s)$ está definida. Definimos
  $$Succ(s)={a(s)|a\in A(s)}$$
- Una función de costo $c:\mathcal{A}\rightarrow\mathbb{R}^+$.
- Un estadio inicial s_{init}.
- Un conjunto de estados finales G.

Solución de un problema de búsqueda

- Una secuencia de operadores $o_0o_1...o_n$ es aplicable en s_0 si y solo si $s_{i+1}=o_i(s_i)$ está definida para todo $i=0,1,...,n$.
- Una secuencia aplicable de operadores $o_0o_1...o_n$ es una solución a un problema de búsqueda si y solo si $s_{i+1}=o_i(s_i)$ está definida para todo $i=0,1,...,n$ y $s_n\in G$. Esto último dice que s_n es un estado objetivo.

PROF

Algoritmo de búsqueda genérico

El siguiente es un algoritmo de búsqueda genérico:
Input: Un problema de búsqueda (S, A, s_{init},G)
Output: Un nodo objetivo.

1. $Open$ es un contenedor vacío.
2. $Closed$ es un conjunto vacío.
3. Inserta s_{init} en $Open$.
4. $parent(s_{init})=null$
5. while $Open$ no está vacío do
6. - $u\leftarrow$ Extraer($Open$)
7. - Inserta u en $Closed$
8. - for each $v\in Succ(u)\setminus(Open\cup Closed)$ do

9. ◦ ▪ $parent(v)=u$
10. ◦ ▪ if $v\in G$ return v
11. ◦ ▪ Inserta v a $Open$

Búsqueda en profundidad (DFS)

- Resulta de implementar a $Open$ como un stack.
- Siempre se extrae el elemento al tope de $Open$.

Búsqueda en amplitud (BFS)

- Resulta de implementar a $Open$ como una cola.
- Siempre se extrae el elemento al principio de $Open$.

Propiedades

Teorema

Si el espacio de estados es finito, entonces DFS con detección de ciclos es completo (es decir, encuentra una solución si existe una).

Teorema

Si el espacio de estados es finito, entonces BFS es completo y óptimo para problemas de búsqueda con constos uniformes.

Tiempo y espacio

Para los siguientes resultados, suponga:

- b: el factor de ramificación promedio.
- p: profundidad a la que se encuentra la solución.
- m: profundidad máxima del árbol de búsqueda (largo de la rama más larga del árbol de búsqueda).

PROF

Teorema

- La memoria usada por DFS es $O(bm)$.
- La memoria usada por BFS es $O(b^p)$.

Teorema

- El tiempo de ejecución de DFS es $O(b^m)$.
- El tiempo de ejecución de BFS es $O(b^p)$.

Profundidad limitada

- Funciona como DFS pero recibe como parámetro un límite l de profundidad para la búsqueda. Se ejecuta DFS sobre el suárbol de brofundidad l del espacio de búsqueda.

Profundización iterativa (Iterative Deepening DFS)

1. $l=1$
2. Realice búsqueda en profundidad con límite l.
3. Si no se encontró solución, incremente l y regrese al paso 2. En caos contrario, retorne la solución encontrada.

Teorema

Profundización iterativa es completo.

Teorema

- El tiempo de ejecución de profundización iterativa es $O(b^p)$.
- El espacio de ejecución de profundización iterativa es $O(bp)$.

Búsqueda informada

¿Qué podemos hacer para mejorar la búsqueda en estos casos?

Búsqueda el mejor primero (*Best-first search*)

De manera intuitiva, este algoritmo:

- Mantiene una lista de $Open$ y $Closed$
- Funciona como DFS, pero
    - Los nodos en $Open$ tienen asociados una calidad.
    - Siempre extrae de $Open$ el nodo de mejor calidad.
    - Un estado sucesor es descartado si está en $Closed$ con mejor o igual calidad.

Función heurística

- En búsqueda informada, usamos una función de estimación del costo de un nodo del árbol de bpusqueda a una solución. La denotamos como
  $$h(n)$$
- En el problema de navegación, si
  $$\Delta x = |x_{obj}-x|, \Delta y = |y_{obj}-y|,$$
  donde (x,y) es la posición actual y (x_{obj},y_{obj}) es el objetivo. La siguiente es una posible heurística (también llama distancia *octile*)
  $$h(x,y)=|\Delta{x}-\Delta{y}|+\sqrt{2}\min{{\Delta{x},\Delta{y}}}$$
  Utilizar únicamente la función heurística conduce a soluciones no óptimas. Para encontrar estas, debemos utilizar una función de costo.

PROF

Incorporando el costo

- Es posible encontrar soluciones óptimas al incorporar el costo incurrido hasta llegar un nodo n.
- Denotamos este costo como $g(n)$

- Luego, podemos ordenar la frontera de búsqueda por la siguiente función: $$f(n)=g(n)+h(n)$$

Algoritmo A*

Input: Un problema de búsqueda (S,A,s_0,G)
Output: Un nodo objetivo

1. for each $s\in\mathcal{S}$ do $g(s)\leftarrow\infty$
2. $Open \leftarrow{{s_0}}$
3. $g(s_0)\leftarrow0; f(s_0) \leftarrow h(s_0)$
4. while $Open\neq\empty$
5. - extrae un nodo u desde $Open$ con el menor $f-score$
6. - if u es objetivo return u
7. - for each $v\in Succ(u)$ do
8. - - insertar v

Procedimiento insertar

1. $cost_v=g(u)+c(u,v)$ // el costo de llegar a v a través de u
2. if $cost_v\geq g(v)$ return // seguimos solo si $cost_v<g(v)$
3. parent(v)$\leftarrow u$
4. $g(v)\leftarrow cost_v$
5. $f(v)\leftarrow g(v)+h(v)$
6. if $v\in Open$ then Recordad $Open$
7. else Insertar v en $Open$

A* y Greedy

- Si usamos $f(n)=h(n)$ en A*, entonces el algoritmo resultante es *greedy best-first search*.
- Los algoritmos ambiciosos encuentran soluciones más rápidamente, sacrificando la calidad de la solución.

PROF

Optimalidad de A*