Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Ayudantía - Search

Matías Greco

IIC2613 - Inteligencia Artificial

Segundo Semestre, 2019

1. Describa cuando una heurística es admisible y cuando es consistente. ¿Que propiedades le otorgan al algoritmo A* usar una heurística que contenga estas características?. Si la heurística es admisible pero no consistente, como podría convertirla en consistente?

2. Demuestre que A*, con una heurística admisible es completo y óptimo.

3. Demuestre que wA*, con una heurística admisible, es w sub-optimo.

4. En algunos problemas como el TSP, es posible implementar A* sin lista de $CLOSE$. Es posible garantizar la no generación de estados duplicados?. Construya el espacio de estados para este problema

5. Encuentre una heurística admisible y consistente para el problema del cubo Rubik's. Intente basarse en como aplicar la distancia de Manhattan en un problema de estas características.

6. A*-early es una implementación de A* que verifica si el nodo es una solución al momento de generarlo. Explique por qué el resultado obtenido por A*-early (tal como fue descrito en el parrafo anterior) no es óptimo. Como lo hace A*-early para corroborar la optimalidad de la solución entregada?.

Algorithm 1: A*-LATE

```
   Input: (Start state s)
1  g(s) ← 0; OPEN ← ∅; CLOSED ← ∅
2  Add s to OPEN with f(s) = h(s)
3  while (OPEN ≠ ∅) do
4      best ← ExtractMin(OPEN)
5      if GoalTest(best)==TRUE then
6          return the lowest-cost path found to best
7      Move best from OPEN to CLOSED
8      for every action A applicable on state best do
9          c ← generate a state by applying A to best
10         g_new ← g(best) + cost(best, c)
11         if c in OPEN ∪ CLOSED then
12             if g(c) ≤ g_new then
13                 continue (duplicate node, goto line 9)
14             Remove c from OPEN and CLOSED
15         g(c) ← g_new
16         Insert c to OPEN with key f(c) = g(c) + h(c)
17 return No solution exists
```

((a)) A*-late

Algorithm 2: A*-EARLY

```
   Input: (Start state s)
1  U ← ∞
2  g(s) ← 0;
3  OPEN ← ∅;
4  CLOSED ← ∅
5  Add s to OPEN with f(s) = h(s)
6  while (OPEN ≠ ∅ and fmin < U) do
7      best ← ExtractMin(OPEN)
8      Move best from OPEN to CLOSED
9      for every action A applicable on state best do
10         c ← generate a state by applying A to best
11         g_new ← g(best) + cost(best, c)
12         if c in OPEN ∪ CLOSED then
13             if g(c) ≤ g_new then
14                 continue (duplicate node, goto line 9)
15             Remove c from OPEN and CLOSED
16         if GoalTest(c)==TRUE then
17             if g_new < U then
18                 U ← g_new
19                 empty-open(U) // optional.
20         g(c) ← g_new
21         f(c) = g(c) + h(c)
22         if f(c) < U then
23             Insert c to OPEN with key f(c)
24 return U and the associated path
```

((b)) A*-early

7. Muestre como quedan la lista de $OPEN$ y $CLOSE$ al ejecutar A* utilizando como heurística la distancia de *Manhattan* en el problema de navegación sobre una grilla de la figura 2.

Figura 1: Pseudocodigo de A*-late (implementación clásica de A*) y A*-early (implementación que verifica si el nodo es goal al generarlo). Fuente: [1]

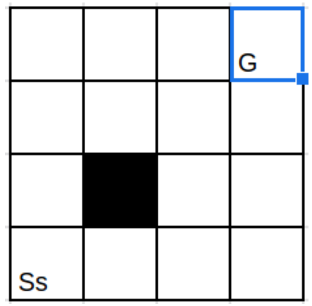

Figura 2: Un problema de búsqueda en grilla

8. Búsqueda bidireccional es un tipo de búsqueda que consiste en intercalar episodios de búsqueda desde atrás hacia adelante y viceversa. Es decir, en los episodios pares expande estados desde el estado inicial, y en los episodios impares expande estados desde el estado objetivo hacia atrás, aplicando las acciones inversas. Un enfoque intuitivo para este tipo de búsqueda, pero que no funciona muy bien, consiste en verificar cuando un camino ya fue generado desde su lado inverso y después unir ambos. Explique por que este enfoque no funciona muy bien y como lo podría mejorar. Utilice la grilla del problema anterior para graficar.

Referencias

[1] Ariel Felner. Position paper: Using early goal test in a. In *Eleventh Annual Symposium on Combinatorial Search*, 2018.

1. Solution

1.-

- Admisibilidad:
  - $h(s) \leq C^*(S, S_g)$
- Consistencia:
  - $h(s_g) = 0$
  - $h(s) \leq c(s, s') + h(s')$]

Cualquier heurística admisible y no consistente puede ser convertida en consistente utilizando la ecuación de Pathmax. $f(s') = max(f(s), f(s'))$

2.- Suponga que de Open se extrae un nodo $S_{g_2} \in G$ (es un estado objetivo) que es de costo sub-óptimo; y sea C^* el costo de la solución óptima.

$$f(S_{g2}) = g(S_{g2}) + h(S_{g2}) \tag{1}$$

Como S_{g2} es sub-óptimo y $h(S_{g_2}) = 0$ (cierto para cualquier heurística admisible).

$$g(S_{g2}) > C* \tag{2}$$

Considerando un nodo $S \notin G$ que está dentro de un camino óptimo. Como $h(s) \leq C^*$ (dado que es admisible)

$$f(s) = g(s) + h(s) \leq C^* \tag{3}$$

Esto demuestra que:

$$f(s) \leq C^* < f(S_{g_2}) \tag{4}$$

Por lo tanto, S_{g_2} no será expandido ya que A* expande siempre el con menor f.

Nota: En esta demostración se utilizó C* para referirse al costo de la solución óptima. En otras demostraciones también podría encontrarlo como $h^*(S_0)$. Recordar que el * simboliza optimalidad.