Práctica Simbólica - SI

David Rodríguez Bacelar y Juan Ramón Pérez García

March 21, 2021

1 Ejercicio 1. Búsqueda Informada

1.1 Búsqueda Avara

Paso	*Frontera*	*Explorados*
1	start(2)	-
2	A(6), B(4), C(6)	start(0)
3	A(6), C(6), E(2)	start(0), B(10)
4	A(6), C(6), D(2), end(0)	start(0), B(10), E(12)
5	A(6), C(6), D(2)	start(0), B(10), E(12), end(18)

1.2 Búsqueda A*

Paso	*Frontera*	*Explorados*
1	start(0+2)	-
2	A(5+6), B(10+4), C(5+6)	start(0)
3	E(9+2), B(10+4), C(5+6)	start(0), A(5)
4	E(9+2), B(9+4), F(11+3)	start(0), A(5), C(5)
5	D(13+2), end(15+0), B(9+4), F(11+3)	start(0), A(5), C(5), E(9)
6	D(13+2), end(15+0), F(11+3)	start(0), A(5), C(5), E(9), B(9)
7	D(13+2), end(14+0)	start(0), A(5), C(5), E(9), B(9), F(11)
8	D(13+2)	start(0), A(5), C(5), E(9), B(9), F(11), end(14)

1.3 Resultados:

A pesar de que la Búsqueda Avara encuentra la solución expandiendo muchos menos nodos que A*, ésta no es óptima. En cambio, A* utiliza una estrategia de búsqueda que, si se apoya en una heurística consistente, logra encotrar el camino óptimo (aunque como vemos también, sacrificando algo más de complejidad temporal y espacial).

2 Ejercicio 2. Búsqueda con Graph Search

1. Distancias:

	Inicio	*Uno*	*Dos*	*Tres*	*Meta*
Inicio	0	8	6	8	10
Uno	8	0	6	8	10
Dos	6	6	0	4	6
Tres	8	8	4	0	2
Meta	10	10	6	2	0

2. Estados:
   - Robot en 'Inicio'. {inicio}
   - Robot en 'Uno' con paquete 1. {Uno(1)}
   - Robot en 'Dos' con paquete 2. {Dos(2)}
   - Robot en 'Tres' con paquete 3. {Tres(3)}
   - Robot en 'Tres' con paquetes 1 y 3. {Tres(1,3)}
   - Robot en 'Dos' con paquetes 1 y 2. {Dos(1,2)}
   - Robot en 'Uno' con paquetes 2 y 1. {Uno(2,1)}
   - Robot en 'Tres' con paquetes 2 y 3. {Tres(2,3)}
   - Robot en 'Uno' con paquetes 3 y 1. {Uno(3,1)}
   - Robot en 'Dos' con paquetes 3 y 2. {Dos(3,2)}
   - Robot en 'Tres' con paquetes 1, 2 y 3. {Tres(1,2,3)}
   - Robot en 'Dos' con paquetes 1, 2 y 3. {Dos(1,2,3)}
   - Robot en 'Uno' con paquetes 1, 2 y 3. {Uno(1,2,3)}
   - Robot en 'Meta'. {meta}

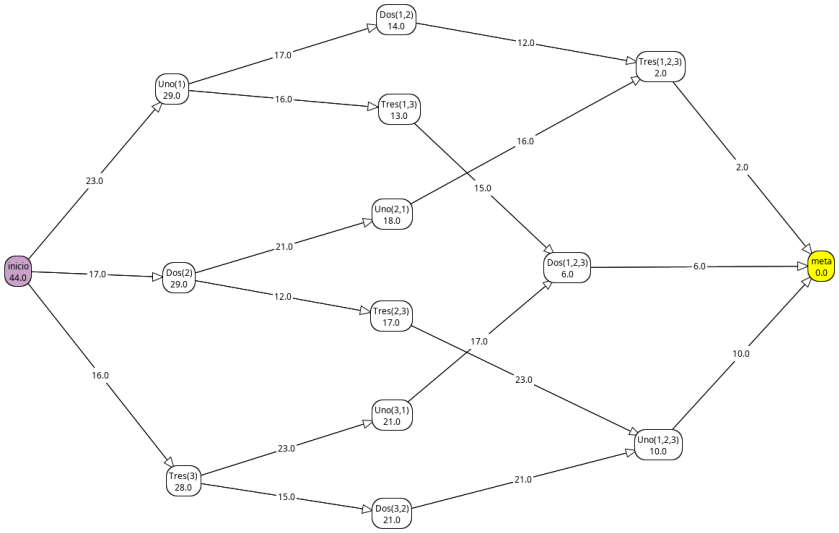
3. Acciones:
   - ◇ Ir a 'Uno' y coger el paquete.
     PreCD: No se han recogido todos los paquetes y tampoco el de la localidad 'Uno'.
   - ◇ Ir a 'Uno' y coger el paquete.
     PreCD: No se han recogido todos los paquetes y tampoco el de la localidad 'Dos'.
   - ◇ Ir a 'Tres' y coger el paquete.
     PreCD: No se han recogido todos los paquetes y tampoco el de la localidad 'Tres'.
   - ◇ Ir a 'Meta'.
     PreCD: Todos los paquetes han sido recogidos.

4. Heurística:

   La heurística utilizada será la distanicia mínima a la meta desde cualquier localidad, sumado al coste de levantar los paquetes restantes. Es una buena heurística ya que siempre nos proporciona costes de camino subestimados 'relajando' el problema; para que el coste del camino real fuese igual al estimado, sería necesario que los paquetes restantes se encontraran en el camino de distancia mínima hasta la meta (el mejor caso), por lo que en otro escenario distinto a ese, el coste sería superior al estimado.

$$h(x) = d(x, meta) + Coste\_de\_levantar\_los\_paquetes\_restantes \tag{1}$$

5. Grafo:



6. Resultados:

   - Depth First:
     - Camino encontrado: inicio → Dos(2) → Tres(2,3) → Uno(1,2,3) → meta
     - Coste del camino: 62
     - Nodos expandidos: 5
   - Breadth First:
     - Camino encontrado: inicio → Dos(2) → Tres(2,3) → Uno(1,2,3) → meta
     - Coste del camino: 62
     - Nodos expandidos: 17
   - Lowest Cost First:
     - Camino encontrado: inicio → Uno(1) → Dos(1,2) → Tres(1,2,3) → meta
     - Coste del camino: 54
     - Nodos expandidos: 16

- Best First:
  - Camino encontrado: inicio → Tres(3) → Dos(3,2) → Uno(1,2,3) → meta
  - Coste del camino: 62
  - Nodos expandidos: 5
- A*:
  - Camino encontrado: inicio → Uno(1) → Dos(1,2) → Tres(1,2,3) → meta
  - Coste del camino: 54
  - Nodos expandidos: 10

(Todas las busquedas han sido realizadas con selección de vecinos por orden alfabético)

7. Conclusiones y discusión:

   El algoritmo que mejores resultados ofrece es *A** ya que, encontrando la solución óptima, expande menos nodos que *Lowest Cost First*. *A** será, por lo general, el mejor algoritmo que garantize la optimalidad (y completitud) expandiendo el menor número de nodos posibles en problemas no muy grandes, con factores de ramificación finitos y con la posibilidad de hallar una heurística consistente (como el nuestro). La ventaja que tienen los algoritmos como *Best First* o o *Depth First* es que hallan también una solución (aunque no óptima) expandiendo muy pocos nodos, ambos solo 5.

   Si variamos mínimamente el problema, los algoritmos como *Breadth First* (que solo garantiza optimalidad cuando los costes de los caminos son constantes), *Depth First* o *Best First*, es muy probable que sigan sin encontrar soluciones óptimas; pero dado que los parámetros de nuestro problema son finitos (factor de ramificación, espacio de estados...) todos seguirían encontrando soluciones no óptimas. En el caso de *Lowest Cost First* y *A**, las soluciones que encontrarían serían igualmente óptimas.