UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Recursive Best First Search[1]

Alfons Juan
Jorge Civera
Albert Sanchis

DSIC

Departamento de Sistemas Informáticos y Computación

[1] Para una correcta visualización, se requiere Acrobat Reader v. 7.0 o superior.

Objetivos

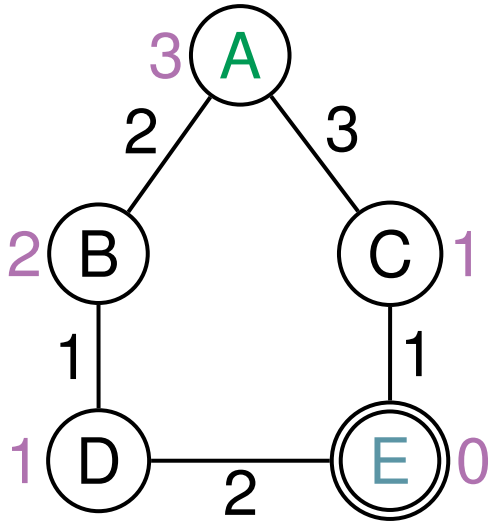
- Aplicar el algoritmo RBFS.
- Construir el árbol de búsqueda RBFS.
- Analizar la optimalidad y complejidad de la búsqueda RBFS.

Índice

1 Introducción

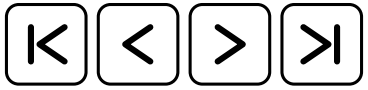
RBFS se basa en búsqueda con backtracking acotada con un valor f, pero, a diferencia de IDA*, garantiza explorar primero-el-mejor para funciones de evaluación no monótonas.

Grafo de estados



Búsqueda

A 0+3	$F_A = 3$	bound = ∞
AB 2+2	$F_B = 4$	
AC 3+1	$F_C = 4$	bound = 4
ABD 3+1	$F_D = 4$	
ABDE 5+0	$F_E = 5$	
ABD	$F_D = 5$	
AB	$F_B = 5$	
AC	$F_C = 4$	bound = 5
ACE 4+0	$F_E = 4$	
ACE sol.!		



RBFS obtiene la cota del segundo-mejor valor f de nodos hermanos en el camino explorado.

2 El algoritmo RBFS (main) [1]

RBFS(G, s', f**)** // G grafo ponderado, s' comienzo, f ***func. eval.***

$P = InitStack(s')$ // Inicializa ***Path*** con el nodo raíz

$b = \infty$ // Cota inicial

$F_{s'} = f_{s'}$ // El valor almacenado es inicializado al valor f

$(F_r, r) =$ **BT**$(G, P, F_{s'}, f, b)$ // Devuelve v. almacenado y est. obj.

if $r \neq$ NULL: **return** P // Si solución, devuelve ***Path*** al objetivo

El algoritmo RBFS (backtracking) [1]

BT(G, P, F_s, f, b) // G grafo, P ***Path***, Valor almacenado $F_{s'}$, f, b cota

$s = Top(P)$ // ***Path***: extraer cima de la pila

if $Goal(s)$: **return** (f_s, s) // Solución encontrada!

$O = InitQueue()$ // ***Open***: cola de prioridad para nodos hijo

for all $(s, n) \in Adjacents(G, s)$ **and** $n \notin P$: // Generando hijos n no en ***Path***

if $f_s < F_s : F_n = max(f_n, F_s)$ // Si s visitado, el hijo hereda el v. almacenado

else: $F_n = f_n$ // En otro caso, el v. almacenado es ***f***

$Push(O, n, F_n)$ // Hijos ordenados en cola de prioridad por v. almacenado

if $EmptyQueue(O)$: **return** (∞, NULL) // No hijos, cota = ∞

while True:

$(n, F_n) = Top(O)$ // Mejor hijo en función del valor almacenado F

if $F_n > b$: **return** (F_n, NULL) // Se excede la cota, backtracking

$(n', F_{n'}) = Top2(O)$ // 2-mejor F o si no existe, entonces $F_{n'} = \infty$

$Push(P, n)$ // Añadir hijo al ***Path*** explorado

$(F_n, r) =$ **BT**$(G, P, F_n, f, min(b, F_{n'}))$ // Recursión con posible nueva cota

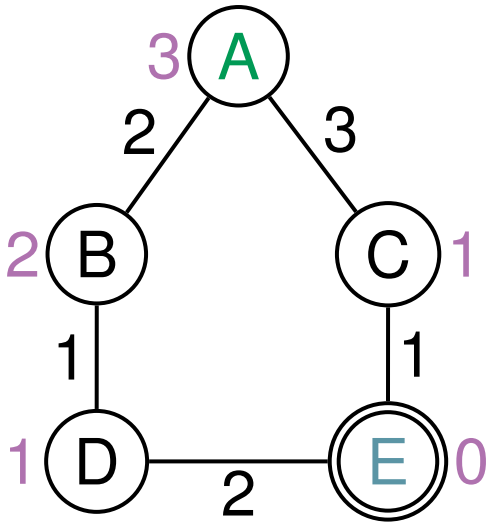
if $r \neq NULL$: **return** (F_n, r) // Si solución, fin recursión sin actualización

$Update(O, n, F_n)$ // Actualizar nodo n en O

$Pop(P)$ // Descartar último hijo del ***Path***

3 Espacio de búsqueda RBFS

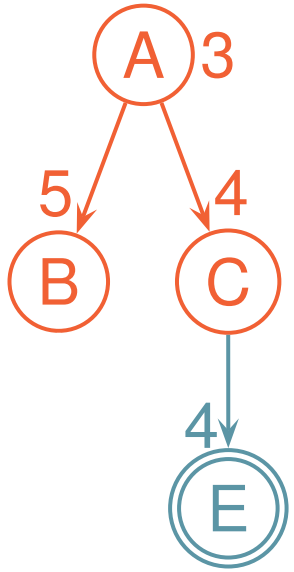
Grafo de estados



RBFS

$b = \infty$

$b = 5$

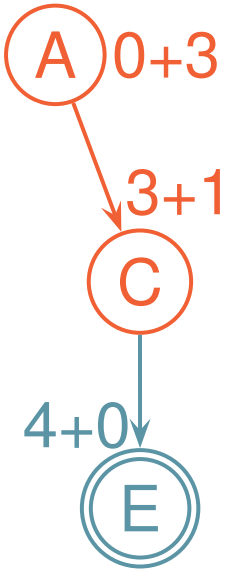


Path: A3C4B5E4

IDA*

$b = 4$

$min = 5$



It 2: A3C4E4

4 Propiedades

- Un nodo ha sido visitado cuando $f_s < F_s$, en otro caso $f_s = F_s$
  - $f_s < F_s$: El hijo hereda el valor almacenado del padre si $f_n < F_s$
  - $f_s = F_s$: El valor almacenado del hijo es f_n en la primera exploración
- La cota se actualiza cuando se entra en la recursión
  - Mínimo entre la cota actual y el 2-mejor valor almacenado en nodos hijo
- F_n es el mínimo valor ***f*** del subárbol expandido por debajo de n
  - F_n se actualiza cuando se sale de la recursión
- Nuevos nodos explorados en orden primero-el-mejor, incluso para funciones ***f*** no monótonas
- Backtracking sólo previene ciclos en el ***Path***

5 Optimalidad y complejidad

- ***Completitud:*** Como A* siempre finaliza en grafos finitos.
- ***Optimalidad:*** Como primero-el-mejor, depende de la función de evaluación.
- ***Complejidad espacial:*** $O(bd)$
- ***Complejidad temporal:*** $O(b^d)$ como IDA*, en la práctica:
  - Un subconjunto de nodos son re-expandidos en cada iteración
  - Es necesaria la cola de prioridad ***Open*** para los hijos de cada nodo
  - Más eficiente en tiempo que IDA*, re-expansión desde el 2-mejor

6 Conclusiones

Hemos estudiado:

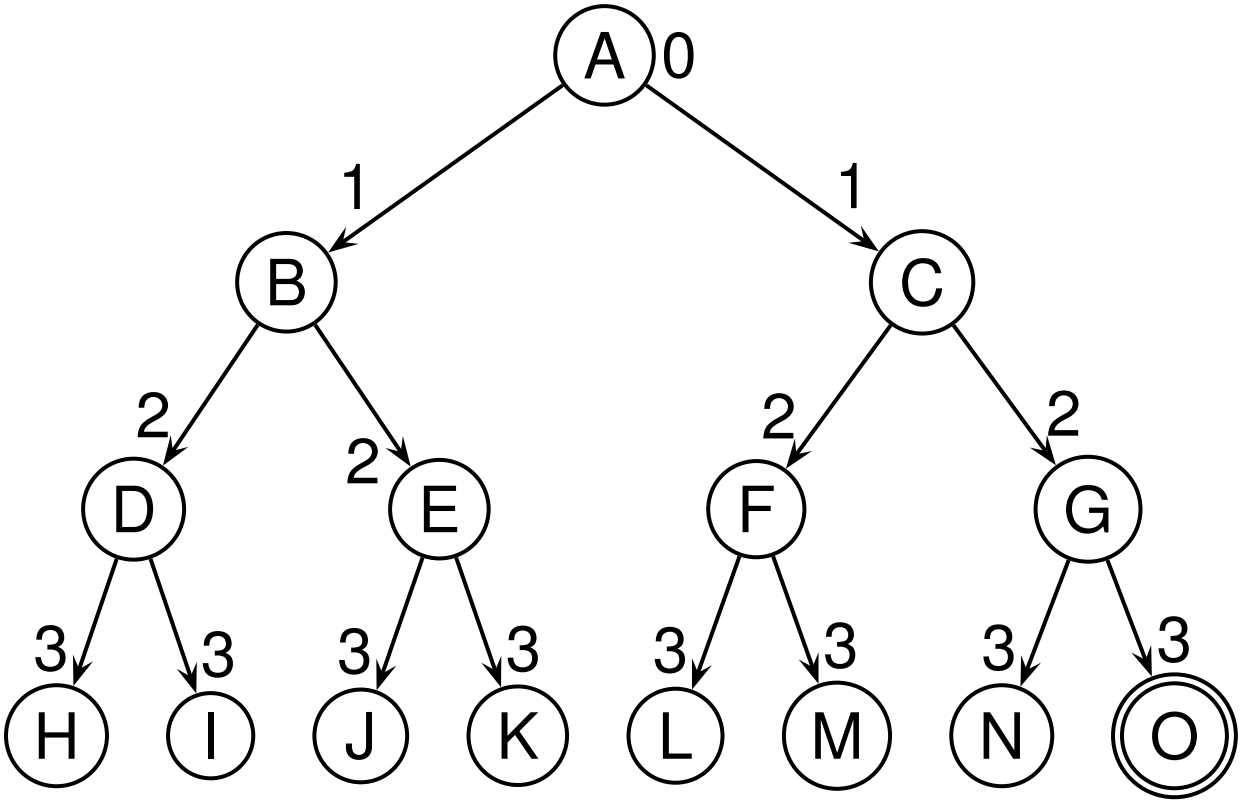
- El algoritmo RBFS.
- El espacio de búsqueda RBFS.
- Propiedades, optimalidad y complejidad de la búsqueda RBFS.

Algunos aspectos a destacar sobre RBFS:

- Completo y óptimo si $f = g + h$ donde h es admisible.
- Coste espacial reducido pero más que IDA*
- Coste temporal depende de la función de evaluación ***f***

Ejercicio RBFS

Realiza una traza de RBFS en el espacio de estados de abajo (f-valor siguiente a cada nodo) y responde a las preguntas del final:



- ¿Máximo número de nodos en memoria?
- ¿Número total de nodos generados?

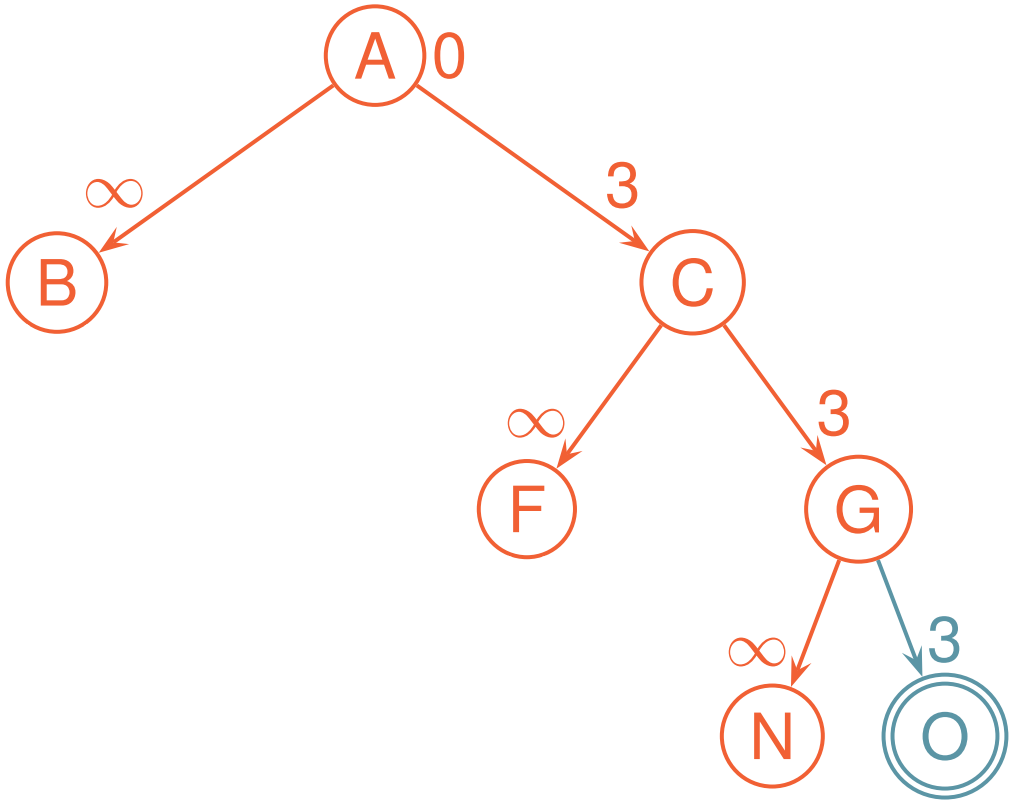
RBFS solución

$b = \infty$

$b = \infty$

$b = \infty$

$b = 3$



Referencias

[1] Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.