UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Iterative Deepening A*[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

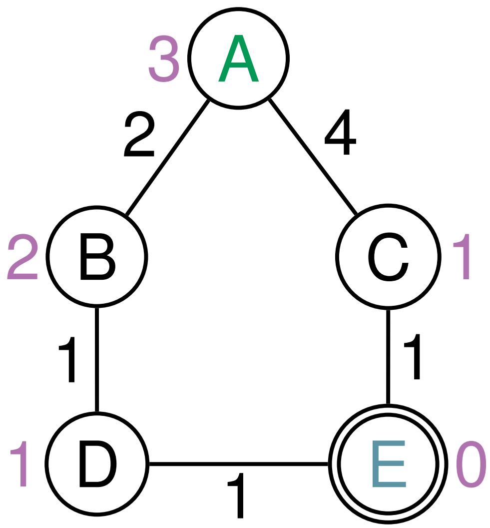
- Aplicar el algoritmo *Iterative Deepening A** (IDA*).
- Construir el árbol de búsqueda IDA*.
- Analizar la optimalidad y complejidad de la búsqueda IDA*.

Índice

1 Introducción

La búsqueda IDA*** está basada en profundización iterativa (con backtracking) utilizando un valor f para acotar la búsqueda en cada iteración en lugar de una profundidad máxima:

Grafo de estados



Búsqueda

A 0+3 bound = 3

AB 2+2
AC 4+1

A 0+3 bound = 4

AB 2+2
ABD 3+1

ABDE 4+0 sol.!

|< < > >|

IDA* calcula la siguiente cota como el valor f mínimo de aquellos nodos que han excedido el valor de la cota actual.

2 El algoritmo IDA* (main) [1]

IDA(G, s', h) // G grafo ponderado, s' comienzo, h ***heuristica***

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

  $b = h(s')$ // Inicializa la cota con $f_{s'} = h(s')$

  while True:

    $(nextb, r) =$ **BT**(G, P, h, b) // *nextb* cota siguiente; *r* estado obj.

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

    if $nextb = \infty$: **return** NULL // no hijos para calcular la sig. cota

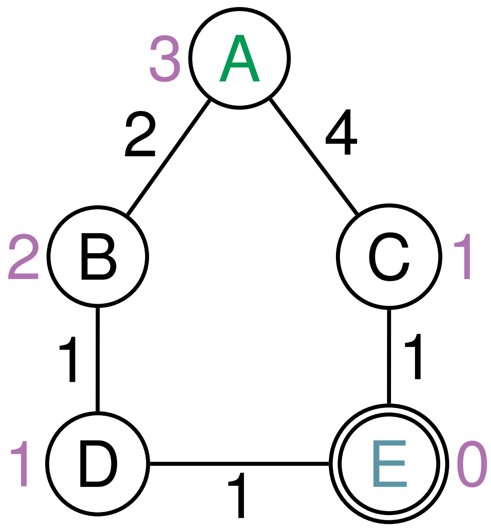
    $b = nextb$ // actualización de la cota para la iteración siguiente

El algoritmo IDA* (backtracking) [1]

BT(G, P, h, b)	// G grafo ponderado, ***Path*** P, h, cota b
$s = Top(P)$	// ***Path***: extrae cima de la pila
$f_s = g_s + h(s)$	// f valor del nodo a explorar
if $f_s > b$: **return** (f_s,NULL)	// b excedida fin para calcular $nextb$
if $Goal(s)$: **return** (f_s, s)	// solución encontrada!
$min = \infty$	// mínimo valor de un hijo f
$n = FirstAdjacent(G, s)$	// generación: n primer hijo de s
while $n \neq$ NULL:	// hijo izquierdo y no en ***Path***
if $n \notin P$:	// n no en ***Path*** para evitar ciclos
$Push(P, n)$	// añadir hijo al ***Path*** explorado
$(nextb, r) =$ BT(G, P, h, b)	// hijo devuelve mín. f y estado sol.
if $r \neq$ NULL: **return** $(nextb, r)$	// si r solución, fin recursión
if $nextb < min$: $min = nextb$	// actualiza valor mín. f
$Pop(P)$	// Descarta último hijo de ***Path***
$n = NextAdjacent(G, s, n)$	// generación: n siguiente hijo de s
return (min, NULL)	// sol. no encontrada, devuelve mínimo f

3 Espacio de búsqueda IDA*

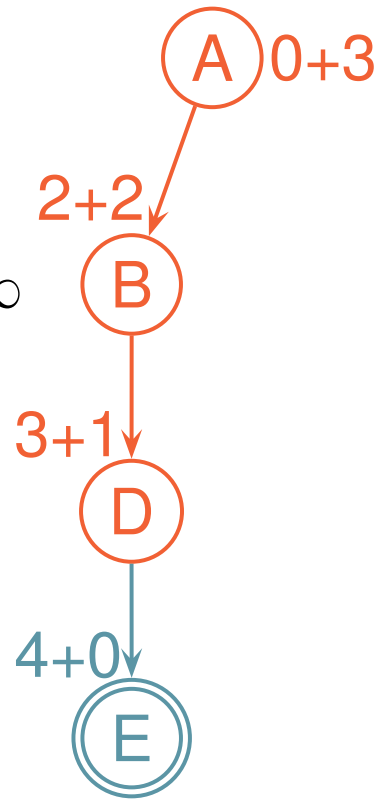
Grafo de estados



IDA*

$b = 4$

$min = \infty$



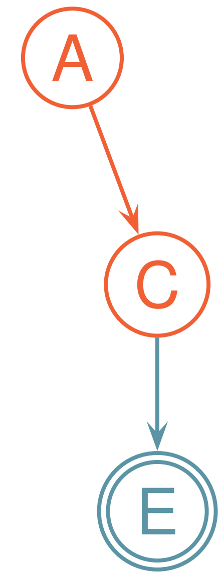
It 2: A3B4D4E4

PI

Prof 0

Prof 1

Prof 2



It 2: ACE

4 Optimalidad y complejidad

- ***Completitud:*** Como A*, siempre finaliza en grafos finitos.
- ***Optimalidad:*** Si h es admisible, IDA* devuelve la solución óptima. IDA* expande nodos en orden creciente de f.
- ***Complejidad espacial:*** Como PI con backtracking, $O(d)$
- ***Complejidad temporal:*** Como A*, $O(b^d)$; en la práctica:
  - Un subconjunto de nodos son re-expandidos en cada iteración
  - Iteraciones dependen del número de nodos con diferente valor f
  - No es necesaria una cola de prioridad en ***Open*** ni lista ***Closed***

5 Conclusiones

Hemos estudiado:

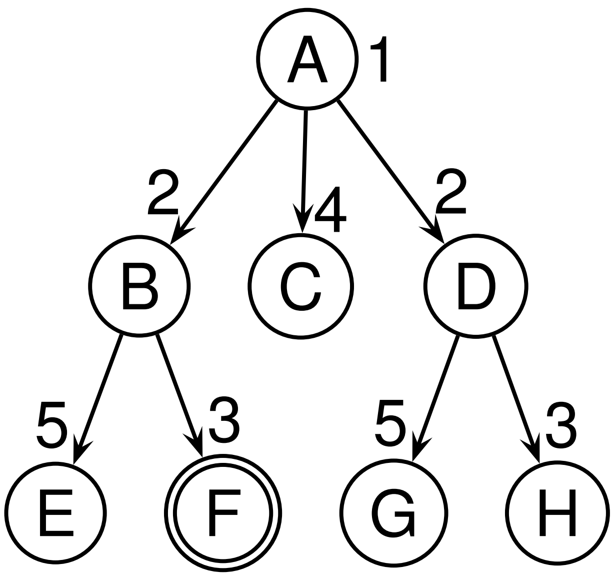
- El algoritmo IDA*.
- El espacio de búsqueda IDA*.
- Optimalidad y complejidad en la búsqueda IDA*.

Algunos aspectos a destacar sobre IDA*:

- Completo y óptimo con costes positivos y h admisible.
- Coste espacial reducido gracias a backtracking.
- El coste temporal depende de la función de evaluación ***f***.

Ejercicio IDA*

valor-f siguiente
a cada nodo



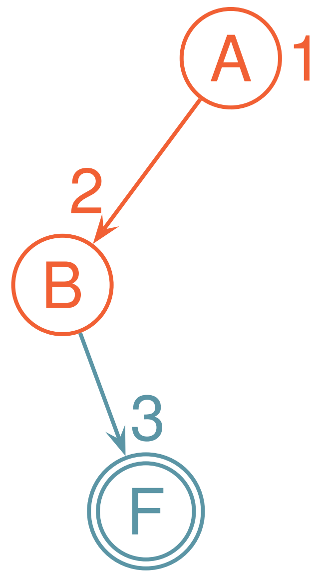
Realiza una traza de IDA* en el espacio de estados de la izquierda y responde a las siguientes preguntas:

- Número de iteraciones hasta encontrar solución?
- Máximo número de nodos en memoria?
- Número total de nodos generados?

Solución IDA*

$b = 3$

$min = \infty$



It 3: nodos generados = 14 sol.!

Referencias

[1] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

PI con backtracking

PI(G, s) // ***Profundidad Iterativa***
  para $m = 0, 1, 2, \ldots$: **si** $(r = \text{BT}(G, s, m)) \neq$ NULL: **retorna** r

BT(G, s, m) // ***Backtracking*** con profundidad máxima m
  si $Objetivo(s)$ **retorna** s // solución encontrada!
  si $m = 0$ **retorna** NULL // profundidad máxima
  $n = PrimerAdyacente(G, s)$ // generación: n primer hijo de s
  mientras $n \neq$ NULL:
    $r = \text{BT}(G, n, m-1)$ // resultado del hijo actual
    si $r \neq$ NULL: **retorna** r // si r es solución, acabamos
    $n = SiguienteAdyacente(G, s, n)$ // generación: n sig. hijo de s
  retorna NULL // ninguna solución encontrada