Iterative Deepening A*[1]

Alfons Juan
Jorge Civera
Albert Sanchis

Departament de Sistemes
Informàtics i Computació

[1] Per a una correcta visualització, es requereix Acrobat Reader v. 7.0 o superior.

Objectius

- Aplicar l'algorisme *Iterative Deepening A** (IDA*).
- Construir l'arbre de cerca IDA*.
- Analitzar l'optimalitat i complexitat de la cerca IDA*.

Índex

1 Introducció 3

2 L'algorisme IDA* 4

3 Espai de cerca IDA* 6

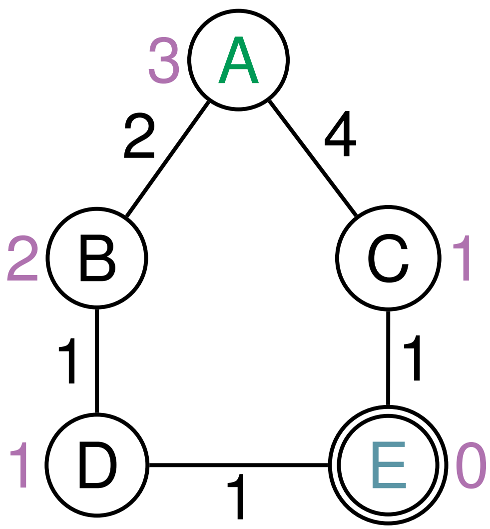
4 Optimalitat i complexitat 7

5 Conclusions 8

1 Introducció

La cerca IDA*** està basada en aprofundiment iteratiu (amb backtracking) utilitzant un valor f per a delimitar la cerca en cada iteració en lloc d'una profunditat màxima:

Graf d'estats



Cerca

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 el següent límit (*bound*) com el valor f mínim d'aquells nodes que han excedit el valor del límit actual.

2 L’algorisme IDA* (main) [1]

IDA(G, s', h**)** // G graf ponderat, s' inici, h ***heuristica***

$P = InitStack(s')$ // Inicialitza ***Path*** amb el node arrel

$b = h(s')$ // Inicialitza el límit amb $f_{s'} = h(s')$

while True:

$(nextb, r) =$ **BT**(G, P, h, b) // *nextb* límit següent; *r* estat obj.

if $r \neq$ NULL: **return** P // si solució, torna ***Path*** al objectiu

if $nextb = \infty$: **return** NULL // no fills per a calcular el seg. límit

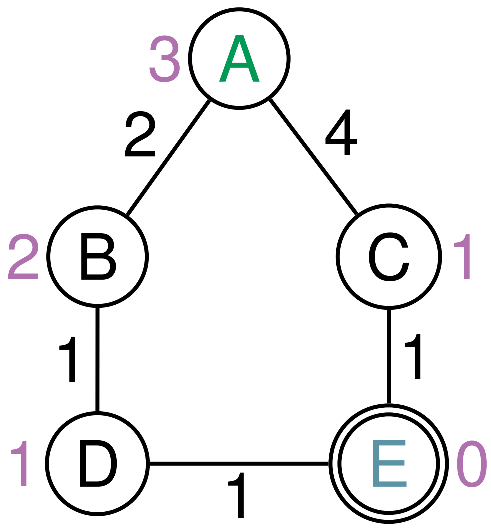
$b = nextb$ // actualització del límit per a la iteració següent

L'algorisme IDA* (backtracking) [1]

```
BT(G, P, h, b)                           // G graf ponderat, P Path, h, b límit
  s = Top(P)                             // Path: extrau cim de la pila
  f_s = g_s + h(s)                       // f valor del node a explorar
  if f_s > b: return (f_s,NULL)          // b excedida fí per a calcular nextb
  if Goal(s): return (f_s, s)            // solució trobada!
  min = ∞                                // mínim valor d'un fill f
  n = FirstAdjacent(G, s)                // generació: n primer fill de s
  while n ≠ NULL:                        // mentre queden fills per explorar
    if n ∉ P:                            // n no en Path per a evitar cicles
      Push(P, n)                         // afegir fill al Path explorat
      (nextb, r) = BT(G, P, h, b)        // fill torna mín. f i estat sol.
      if r ≠ NULL: return (nextb, r)     // si r solució, fí recursió
      if nextb < min: min = nextb        // actualitza valor mín. f
      Pop(P)                             // Descarta últim fill de Path
    n = NextAdjacent(G, s, n)            // generació: n següent fill de s
  return (min, NULL)                     // sol. no trobada, torna mínim f
```

3 Espai de cerca IDA*

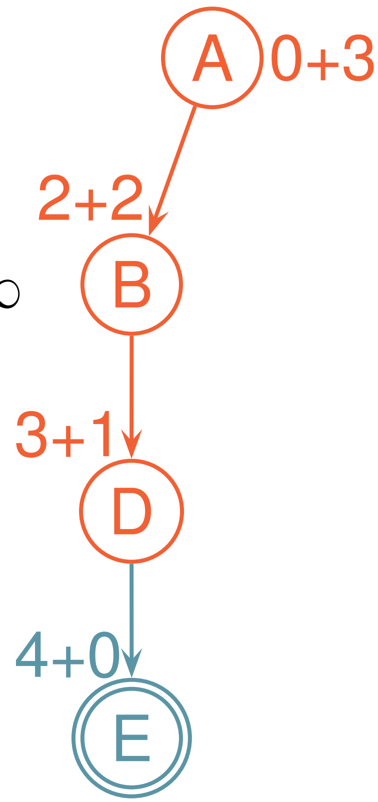
Graf d'estats



*IDA**

$b = 4$

$min = \infty$



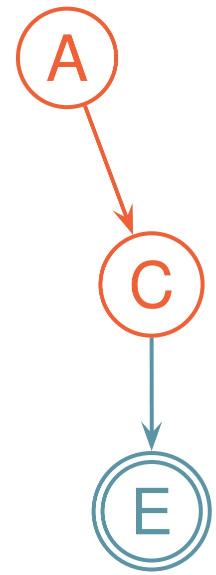
It 2: A3B4D4E4

PI

Prof 0

Prof 1

Prof 2



It 2: ACE

4 Optimalitat i complexitat

- ***Completesa:*** Com A*, sempre finalitza en grafs finits.
- ***Optimalitat:*** Si h és admissible, IDA* retorna la solució òptima. IDA* expandeix nodes en ordre creixent de f.
- ***Complexitat espacial:*** Com PI amb backtracking, $O(d)$
- ***Complexitat temporal:*** Com A*, $O(b^d)$; en la pràctica:
  - Un subconjunt de nodes són re-expandits en cada iteració
  - Iteracions depenen del nombre de nodes amb diferent valor f
  - No és necessària una cua de prioritat en ***Open*** ni llista ***Closed***

5 Conclusions

Hem estudiat:

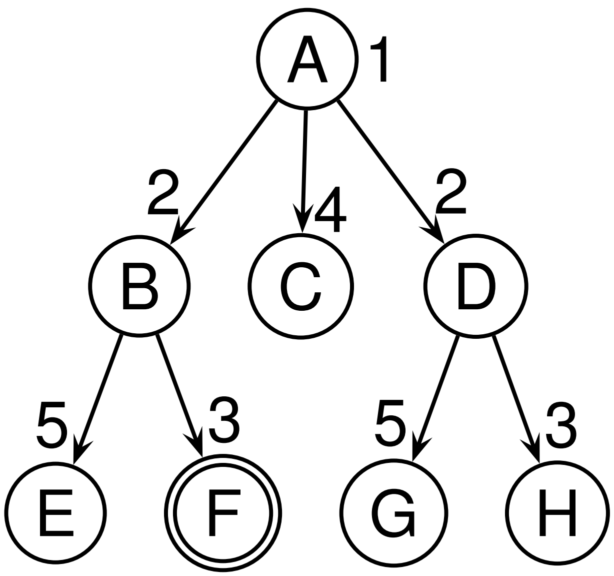
- L'algorisme IDA*.
- L'espai de cerca IDA*.
- Optimalitat i complexitat en la cerca IDA*.

Alguns aspectes a destacar sobre IDA*:

- Complet i òptim amb costos positius i h admissible.
- Cost espacial reduït gràcies a backtracking.
- El cost temporal depèn de la funció d'avaluació ***f***.

Exercici IDA*

valor-f següent
a cada node



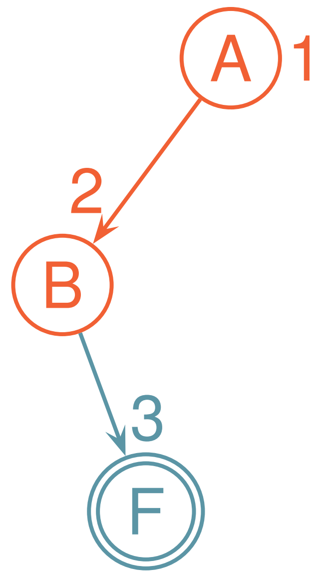
Realitza una traça de IDA* en l'espai d'estats de l'esquerra i respon a les següents preguntes:

- Nombre d'iteracions fins a trobar solució?
- Màxim nombre de nodes en memòria?
- Nombre total de nodes generats?

Solució IDA*

$b = 3$

$min = \infty$



It 3: nodes generats = 14 sol.!

Referències

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

PI amb backtracking

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

BT(G,s,m) // ***Backtracking*** amb profunditat màxima m
  si $Objectiu(s)$ **retorna** s // solució trobada!
  si $m=0$ **retorna** NULL // profunditat màxima
  $n = PrimerAdjacent(G,s)$ // generació: n primer fill d's
  mentre $n \neq \text{NULL}$:
    $r = \text{BT}(G,n,m-1)$ // resultat del fill actual
    si $r \neq \text{NULL}$: **retorna** r // si r és solució, acabem
    $n = SegüentAdjacent(G,s,n)$ // generació: n següent fill d's
  retorna NULL // cap solució trobada