Búsqueda Heurística V

Pedro Meseguer
IIIA-CSIC
Bellaterra, Spain
pedro@iiia.csic.es

Algoritmos

LDS: *limited discrepancy search*

ILDS: *improved limited discrepancy search*

DDS: *depth-bounded discrepancy search*

IDFS: *interleaved depth-first search*

Nodos buenos y malos

- Suponemos:
  - árbol binario *depth-first search*
  - finito, soluciones a profundidad *d*
- Nodo interior:
  - **bueno**: si hay solución en sus desdendientes
  - **malo**: si no hay solución entre sus descendientes
- Papel de la heurística: al generar sucesores, poner nodos buenos a la izquierda de los malos

Errores heurísticos

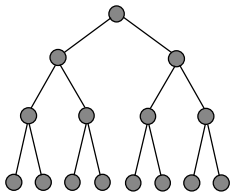
- Error heurístico: cuando *h* ordena un nodo **malo** antes de uno **bueno**
- Depth-first search:
  - cuando entra en un subárbol, no sale de él hasta que lo agota
  - visita subárboles de izquierda a derecha
  - DFS no tiene defensa contra errores de la heurística

Discrepancia

- Árbol de búsqueda, sucesores ordenados por *h*
- En cada nodo interno:
  - sucesor izquierda: recomendado por *h*
  - sucesor derecha: contra *h* → ***discrepancia***
- Discrepancia: ir contra *h* (moverse a rama derecha)
- Si *h* es razonablemente buena, la solución estará a profundidad *d* en ramas con pocas discrepancias

Idea de LDS

- Idea: visitar las ramas por número de discrepancias creciente



- En iteración *i*, se visitan todas las ramas con número de discrepancias ≤ *i*

LDS

LDS(*node*)

for *x* = 0 to maximum number of discrepancies **do**

result ← OLDS(*node*, *x*)

if *result* = *success* **then return** *result*

return *failure*

OLDS(*node*, *k*)

if *node* is a leaf **then return** *success* or *failure*

if (*k* = 0) **then return** OLDS(left-child(*node*), 0)

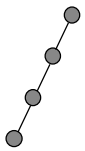
else *result* = OLDS(right-child(*node*), *k*-1)

if *result* = *success* **then return** *result*

else return OLDS(left-child(*s*),*k*)

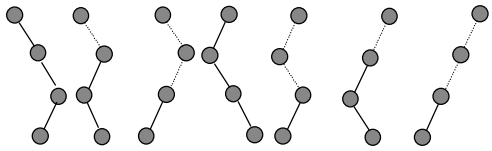
Ejecución LDS

iter 0



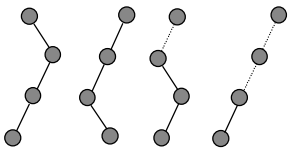
0

iter 2



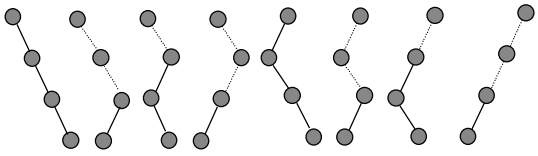
5 6 7 8 9 10 11

iter 1



1 2 3 4

iter 3



12 13 14 15 16 17 18 19

Motivación ILDS

- LDS repite trabajo: en iteración *i* visita todas las ramas con número de discrepancias ≤ *i*
- Mejora: en iteración *i* ILDS visita todas las ramas con número de discrepancias = *i*
- ILDS:
  - no repite trabajo en nodos hoja
  - si repite trabajo en nodos interiores

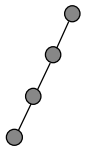
ILDS

```
LDS(node)
  for x = 0 to maximum number of discrepancies do
    result ← ILDS(node, x, max-depth)
    if result = success then return result
  return failure

ILDS(node, k, depth)
  if node is a leaf then return success or failure
  if depth > k then
      result ← ILDS(left-child(node), k, depth-1)
      if result = success then return result
  if (k > 0) return ILDS(right-child(node), k-1, depth-1)
  else return failure
```

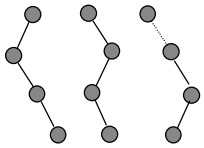
Ejecución ILDS

iter 0



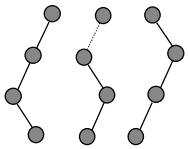
0

iter 2



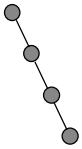
4 5 6

iter 1



1 2 3

iter 3



7

Motivación DDS

- LDS / ILDS: dan la misma importancia a discrepancias arriba o abajo del árbol
- Heurística: tiende a estar más informada abajo en el árbol
- Idea DDS:
  - algoritmo iterativo
  - en iteración *i*, permitimos discrepancias hasta nivel *i*
  - si no encontramos solución, $i \leftarrow i+1$

DDS

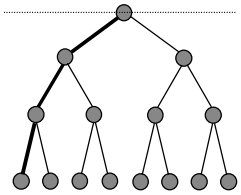
```
DDS
    k ← 0
    repeat
      <goal, depth> ← PROBE(root, k)
      k ← k + 1
    until goal or k > depth
    return goal
```


DDS: PROBE

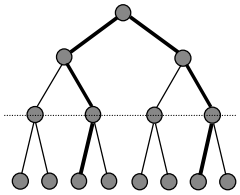
```
PROBE(node, k)
   if leaf then return success or failure
   if k = 0 then <goal, depth> ← PROBE(left-child(node), 0)
                 return <goal, depth+1>
   if k = 1 then <goal, depth> ← PROBE(right-child(node), 0)
                 return <goal, depth+1>
   if k > 1 then <goal1, depth1> ← PROBE(left-child(node), k-1)
        if goal1 then return <goal1, depth+1>
        else <goal2, depth2> ← PROBE(right-child(node), k-1)
             return <goal2, max (depth1, depth2) + 1>
```

Ejecución DDS

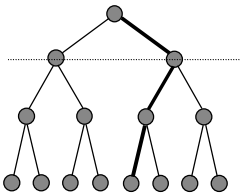
iter 0



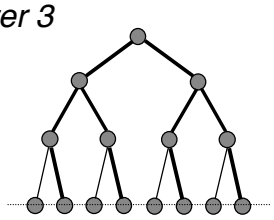
iter 2



iter 1



iter 3



Pure IDFS

if *s* is a leaf **then return** *success* or *failure*
if *s* is not expanded **then** generate *successors*(*s*)
if *successors*(*s*) is not empty **then** (3)
  s' ← extract-first(*sucessors*(*s*));
  result ← IDFS(s')
  case *result* **of**
    continue: add-last(successors(s), s')
    success: **return** *result*
    failure: do nothing
if *successors*(*s*) is empty **then return** *failure*
else if s is the initial node **then goto** 3
else return *continue*

Limited IDFS

if *s* is not expanded **then**
generate *successors*(*s*); *active* (*s*) ← extract-k-first(*successors*(*s*));
if *active*(*s*) is not empty **then**
s' ← extract-first(*active*(*s*))
if *level*(*s'*) + 1 is parallel **then** *result* ← IDFS(s')
else *result* ← IDFS-seq(s´)
case *result* **of**
continue: add-last(active(s),s')
success: **return** *result*
failure: **if** *successors*(*s*) is not empty **then**
add-last(*active*(*s*), extract-first(*successors*(*s*))
if *active*(*s*) is empty **then return** *failure*
else if *s* is the initial node **then goto** 3
else return *continue*

IDFS-seq

if *s* is a leaf **then return** *success* or *failure*
if *s* is not expanded **then** generate *successors*(*s*)
if *successors*(*s*) is not empty **then**
s' ← extract-first(*sucessors*(*s*));
result ← IDFS-seq(s')
case *result* **of**
continue: add-first(successors(s), s')
success: **return** *result*
failure: do nothing
if *successors*(*s*) is empty **then return** *failure*
else return *continue*