Uninformed Search strategies

AIMA sections 3.4

Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

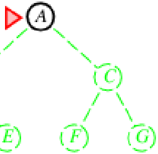
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Breadth-first search

Expand **shallowest** unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at the end

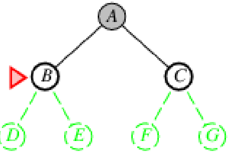


Breadth-first search

Expand **shallowest** unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at the end

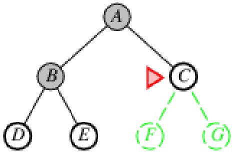


Breadth-first search

Expand **shallowest** unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at the end

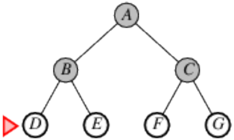


Breadth-first search

Expand **shallowest** unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at the end



Breadth-First Search Algorithm


```
function BFS( problem) returns a solution, or failure
  node ← node with State=problem.Initial-State,Path-Cost=0
  if problem.Goal-Test(node.State) then return node
  explored ← empty set
  frontier ← FIFO queue with node as the only element
  loop do
    if frontier is empty then return failure
    node ← Pop(frontier)
    add node.State to explored
    for each action in problem.Actions(node.State) do
      child ← Child-Node(problem,node,action)
      if child.State is not in explored or frontier then
        if problem.Goal-Test(child.State) then return child
        frontier ← Insert(child)
      end if
    end for
  end loop
```

Properties of breadth-first search

Complete??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time??

Properties of breadth-first search

Complete?? Yes (if b is finite)
Time?? $b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exp. in d
Space??

Properties of breadth-first search

Complete?? Yes (if b is finite)
Time?? $b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exp. in d
Space?? $O(b^d)$ (keeps every node in memory)
Optimal??

Properties of breadth-first search

Complete?? Yes (if b is finite)
Time?? $b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exp. in d
Space?? $O(b^d)$ (keeps every node in memory)
Optimal?? Yes (if cost = 1 per step); not optimal in general

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exp. in d

Space?? $O(b^d)$ (keeps every node in memory)

Optimal?? Yes (if cost = 1 per step); not optimal in general

Space is the big problem; can easily generate nodes at 100MB/sec

so 24hrs = 8640GB.

Uniform cost search

Expand least-cost unexpanded node (i.e., minimum step cost)

Implementation:

frontier = queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

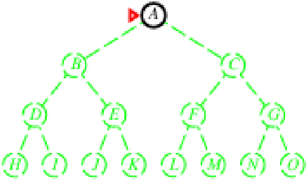
Optimal?? Yes—nodes expanded in increasing order of $g(n)$

Depth-first search

Expand deepest unexpanded node

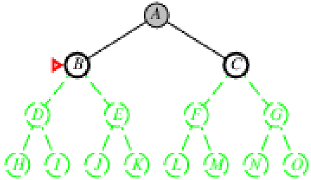
Implementation:

frontier = LIFO queue, i.e., put successors at front



Depth-first search

Expand deepest unexpanded node
Implementation:
  frontier = LIFO queue, i.e., put successors at front

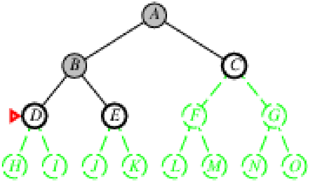


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

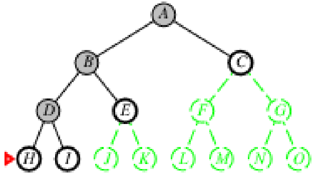


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

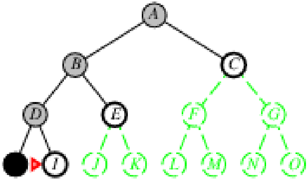


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

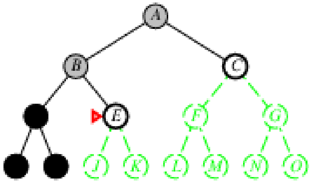


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

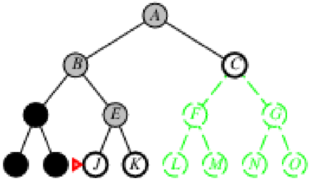


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

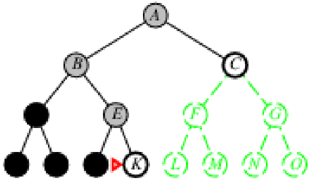


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

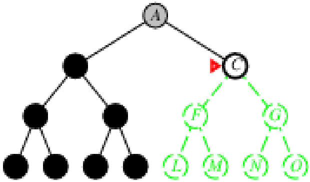


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

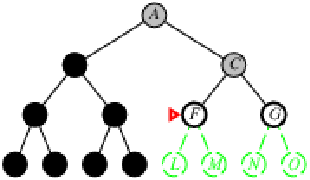


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

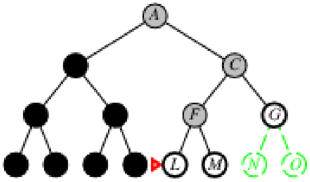


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front

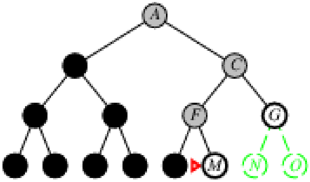


Depth-first search

Expand deepest unexpanded node

Implementation:

frontier = LIFO queue, i.e., put successors at front



Properties of depth-first search

Complete??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal?? No!

Depth-limited search

DFS + depth limit *l*: nodes at depth *l* have no successors

Recursive implementation:

```
function DLS(problem, limit) returns soln/fail/cutoff
    R-DLS(Make-Node(problem.Initial-State), problem, limit)

function R-DLS(node, problem, limit) returns soln/fail/cutoff
    if problem.Goal-Test(node.State) then return node
    else if limit = 0 then return cutoff
    else
        cutoff-occurred? ← false
        for each action in problem.Actions(node.State) do
            child ← Child-Node(problem, node, action)
            result ← R-DLS(child, problem, limit-1)
            if result = cutoff then cutoff-occurred? ← true
            else if result ≠ failure then return result
        end for
       if cutoff-occurred? then return cutoff else return failure
    end else
```

Iterative deepening search

```
function IDS(problem) returns a solution
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DLS(problem, depth)
        if result ≠ cutoff then return result
    end
```

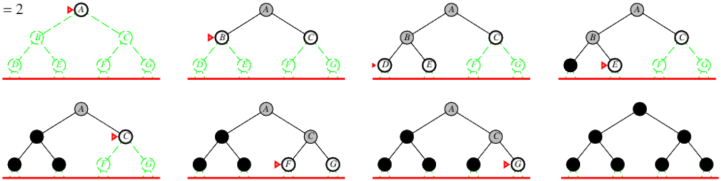
Iterative deepening search

Limit = 0

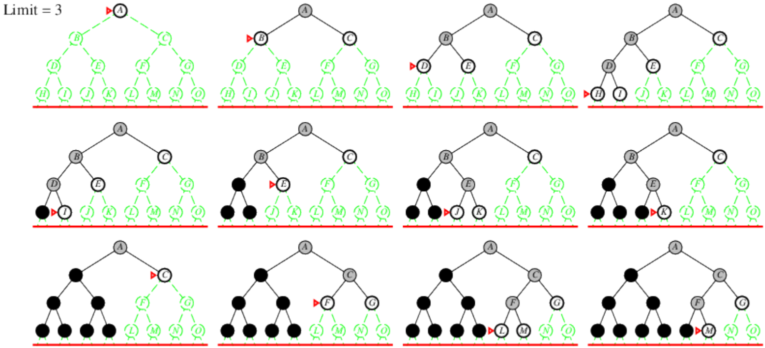
Iterative deepening search

Iterative deepening search

Limit = 2



Iterative deepening search



Properties of iterative deepening search

Complete??

Properties of iterative deepening search

Complete?? Yes

Time??

Properties of iterative deepening search

Complete?? Yes

Time?? $db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space??

Properties of iterative deepening search

Complete?? Yes

Time?? $db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal??

Properties of iterative deepening search

Complete?? Yes

Time?? $db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost = 1

Can be modified to explore uniform-cost tree

BFS Vs IDS

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$\begin{aligned} N(\text{IDS}) &= 50 + 400 + 3,000 + 20,000 + 100,000 \\ &= 123,450 \\ N(\text{BFS}) &= 10 + 100 + 1,000 + 10,000 + 100,000 \\ &= 111,101 \end{aligned}$$

IDS repeats dome nodes but it does not do much worse than BFS because complexity is dominated by exponential growth of nodes.

Summary of algorithms

◇ Considering tree-search versions

Criterion	BF	UC	DF	DL	ID
Complete?	Yes*	Yes*,†	No	Yes*, if $l \geq d$	Yes*
Time	b^d	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^d	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	Yes*, if $l \geq d$	Yes*

*: complete if branching factor is finite

†: complete if step cost is $\geq \epsilon$

⋆: optimal if step costs are all identical

Summary

◇ Variety of uninformed search strategies
◇ Iterative deepening search uses only linear space and not much more time than other uninformed algorithms
◇ Graph search can be exponentially more efficient than tree search

Exercise: Search Space Dimension

BFS vs IDS

Assume: i) a well balanced search tree; ii) the goal state is the last one to be expanded in its level (e.g., the rightmost).

$\diamondsuit$ if the branching factor is 3, the shallowest goal state is at depth 3 (root has depth 0) and we proceed **breadth first** how many nodes are generated ?

$\diamondsuit$ if the branching factor is 3, the shallowest goal state is at depth 3 (root has depth 0) we proceed with an **iterative deepening** approach, how many nodes are generated ?

Exercise: formalizing and solving problem through search

The Wolf Sheep and Cabbage Problem

A man owns a wolf, a sheep and a cabbage: He is on a river bank with a boat that can carry him with only one of his goodies at a time.
The man wants to reach the other bank with his wolf, sheep and cabbage, but he knows that wolves eat sheeps, and sheeps eat cabbages, so he cannot leave them alone on a bank.

- ◇ Formalize the WSC problem as a search problem
- ◇ Use breadth first to find a solution

Exercise: formalizing and solving problem through search

The Missionaries and Cannibals

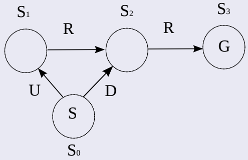
Three missionaries and three cannibals are on the same river bank, and want to cross it.
They have a boat that can carry two people at most. Cannibals should never outnumber missionaries, on any bank, as they could eat them.

- ◇ Formalize the MC problem as a search problem
- ◇ Give a solution

Exercise: Optimality for Graph Search

Differences between different search strategies

Consider the state space graph in the figure, all moves cost 1.



Answer to the following questions:

- State whether a Graph Search version of BFS would always return the optimal solution for this problem, if not provide an execution where this is not the case.
- State whether a Graph Search version of IDS would always return the optimal solution for this problem, if not provide an execution where this is not the case.