

Search Strategies

Uninformed/Blind Search

- Breadth First Search
- Depth First Search
- Depth Limited Search
- Bidirectional Search

Informed/Heuristic Search

- Hill Climbing Search (Improvements)
- A* Algorithm

Measuring problem-Solving performance

What makes one search scheme better than another?

Completeness: Guarantee to find a solution?

Time complexity: How long is it to find a sol. (# of nodes)?

Optimality: Does the strategy find the shortest path (note some books use least cost)?

Space complexity: How much memory is needed (max. # of nodes in memory)?

Notations

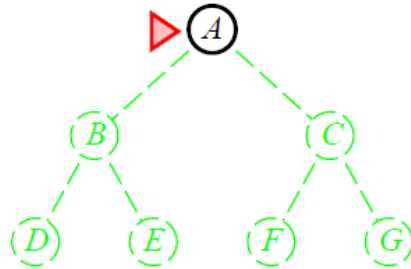
- **b**: Branching Factor that is maximum number of successors of any node
- **d** : depth of the least cost solution
- **C*** : path cost of the optimal solution
- **m** : maximum depth of the state space

Breadth First Search

- Simple Strategy
- The root is expanded first, Then all its successors, Then all their successors
- At a given depth, All nodes are expanded.
- With branching factor b , at level d , we have $1+b+b^2+b^3+\dots+b^d + b(b^d - 1) = O(b^{d+1})$ Nodes
- At level 12 with branching factor 10, we have 10^{13} nodes
- Space Problem !

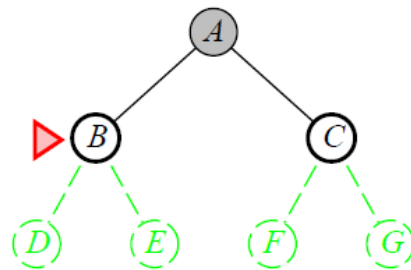
Breadth First Search

- Expand the shallowest node



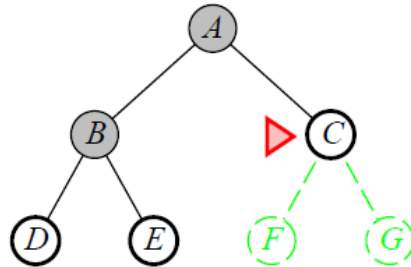
Breadth First Search

- Expand the shallowest node



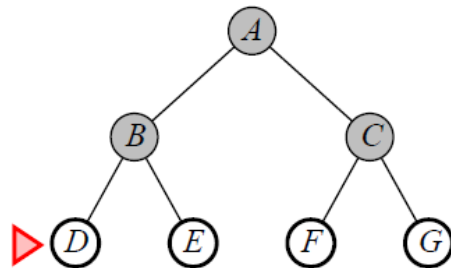
Breadth First Search

- Expand the shallowest node



Breadth First Search

- Expand the shallowest node



BFS

Completeness?

Yes, if solution exists, there is a guarantee to find it

Time complexity?

$O(b^{d+1})$

Space complexity?

$O(b^{d+1})$: keeps every node in memory

Optimality?

Yes : finds shortest path

Remark:

If the definition of optimality is to find lowest cost path then BFS is not optimal

Bidirectional Search

BFS in both directions

How could this help?

b^{d+1} vs $2b^{(d+1)/2}$

- Can reduce time complexity,
- Not always applicable
- May require lots of space
- Hard to implement

Bidirectional Search

Completeness?

Yes, if solution exists, there is a guarantee to find it

Time complexity?

$O(b^{(d+1)/2})$, b is branching factor, d is least cost to goal

Space complexity?

$O(b^{(d+1)/2})$

Optimality?

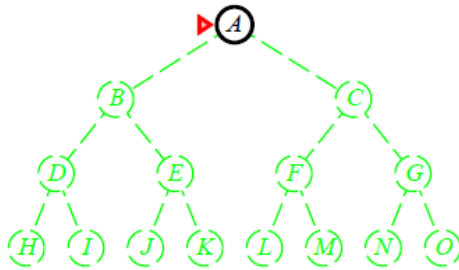
yes

Depth First Search

- Always expand deepest node in the fringe of the tree.
- Modest memory requirement, stores only single path from root to leaf.
- With branching factor b , at level d , we store only $bm+1$ i.e. $O(bm)$
- It may stuck in an infinite path and never finds solution

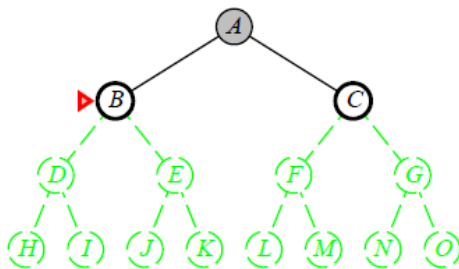
Depth First Search

- Expand deepest unexpanded node



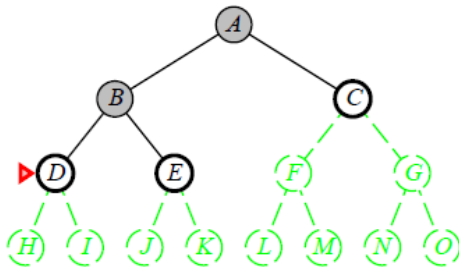
Depth First Search

- Expand deepest unexpanded node



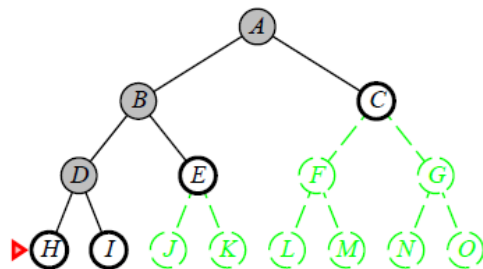
Depth First Search

- Expand deepest unexpanded node



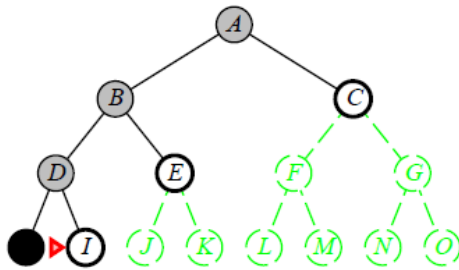
Depth First Search

- Expand deepest unexpanded node



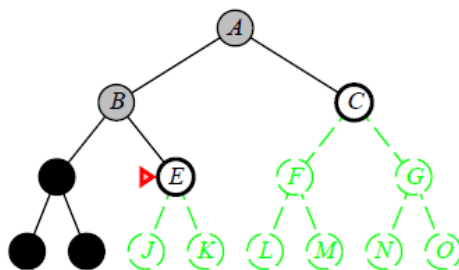
Depth First Search

- Expand deepest unexpanded node



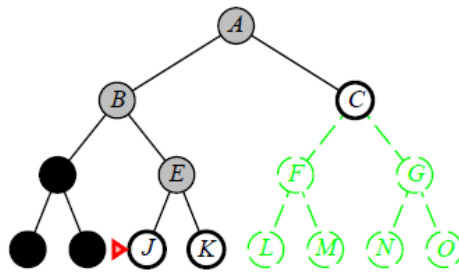
Depth First Search

- Expand deepest unexpanded node



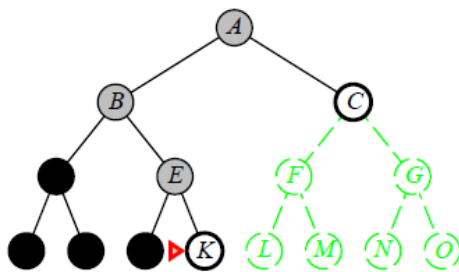
Depth First Search

- Expand deepest unexpanded node



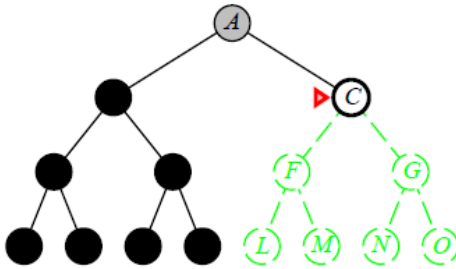
Depth First Search

- Expand deepest unexpanded node



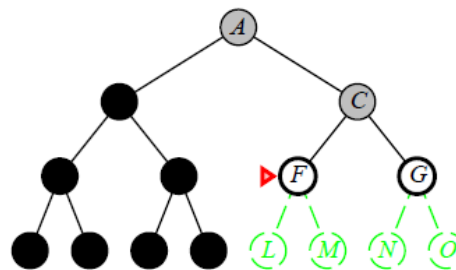
Depth First Search

- Expand deepest unexpanded node



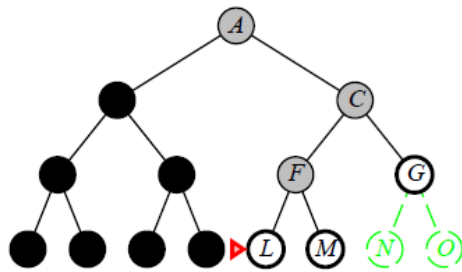
Depth First Search

- Expand deepest unexpanded node



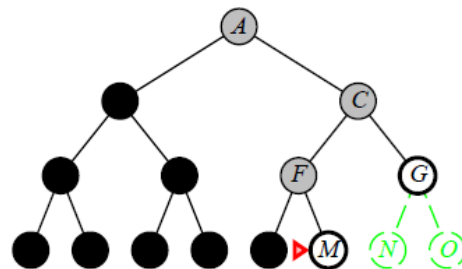
Depth First Search

- Expand deepest unexpanded node



Depth First Search

- Expand deepest unexpanded node



DFS

Completeness?

No, fails in infinite depth spaces or spaces with loops

Yes, assuming state space finite.

Time complexity?

$O(b^m)$, terrible if m is much bigger than d . can do well if lots of goals

Space complexity?

$O(bm)$, i.e. linear

Optimality?

No may find a solution with long path

Depth-limited Search

Put a limit to the level of the tree

DFS, only expand nodes depth $\leq L$.

Completeness?

No, if $L < d$.

Time complexity?

$O(b^L)$

Space complexity?

$O(bL)$

Optimality?

No

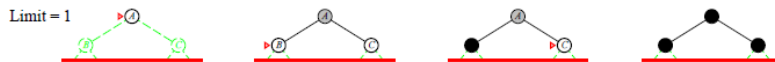
Iterative Deepening

- Calls depth-limited search with increasing limits until goal is found



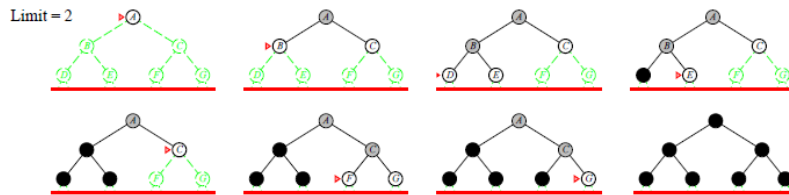
Iterative Deepening

- Calls depth-limited search with increasing limits until goal is found



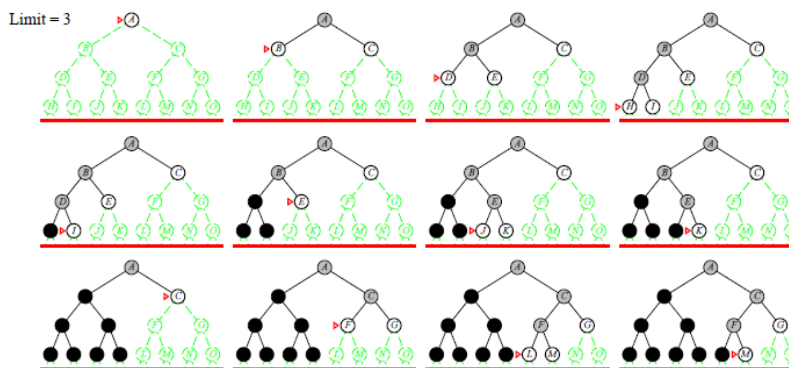
Iterative Deepening

- Calls depth-limited search with increasing limits until goal is found



Iterative Deepening

- Calls depth-limited search with increasing limits until goal is found



Iterative Deepening

- Completeness?

Yes.

Time complexity?

$$O(b^d) = (d+1) b^0 + db^1 + \dots + b^d$$

Space complexity?

$$O(bd)$$

Optimality?

Yes; if looking for shortest path

Remark: IDS performs much faster than BFS:

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

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$

$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

Remarks

- BFS works as a **queue**. Pick the leftmost element of the open list, evaluate it and add its children to the end of the list, FIFO
- DFS works as a **stack**. Pick the leftmost element of the open list, evaluate it and add its children to the beginning of the list, LIFO

Informed Search

- Blind search - no notion concept of the "right direction"
 - can only recognize goal once it's achieved
- Heuristic search – we have rough idea of how good various states are, and use this knowledge to guide our search
- Can find solutions more efficient than uninformed
- General approach is *best-first-search*
- A node is selected based on an *evaluation function $f(n)$*
- A node that **seems** to be best is picked and it may not be the actual best

Best First Search

- **The Idea:**
 - use an *evaluation function* for each node... estimate of "desirability"
 - Expand most desirable unexpanded node

Implementation

- **Fringe:** is a queue sorted in decreasing order of desirability

Special cases

Greedy

A*

Cost function $f(n)$

- A function f is maintained for each node
- $f(n) = g(n) + h(n)$, n is the node in the open list
- “Node chosen” for expansion is the one with least f value
- $g(n)$ is the cost from root S to node n
- $h(n)$ is the estimated cost from node n to a goal
- For BFS: $f = 0$,
- For DFS: $f = 0$,
- For greedy $g = 0$

Greedy search

- Expands a node it sees closest to the goal
- $f(n) = h(n)$
- Resembles DFS in that it prefers to follow a single path all the way to the goal
- Also suffers from the same defects of DFS, it may get stuck in a loop i.e. not complete As well as it is not optimal.

Hill climbing

This is a *greedy* algorithm
 Expands a node it sees closest to a goal

$$f(n) = h(n)$$

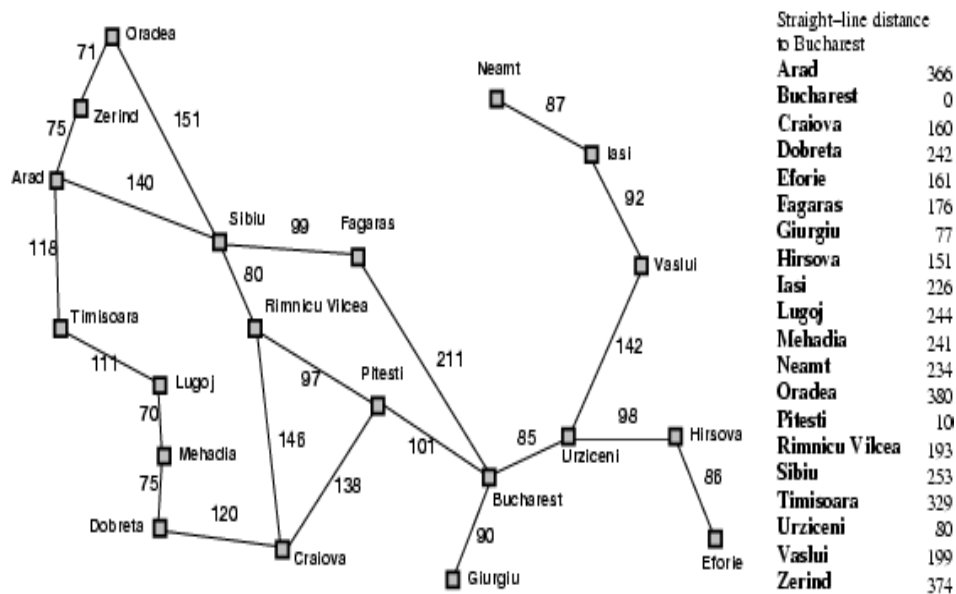
The algorithm

select a heuristic function;
 set C, the current node, to the **highest-valued** initial node;

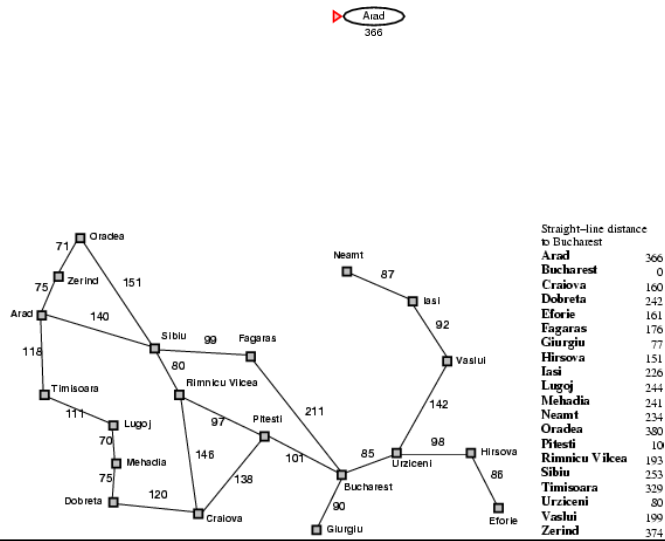
Loop until success or no more children(fail)

select N, the **highest-value** child of C;
 return C if its value is better than the value of N;

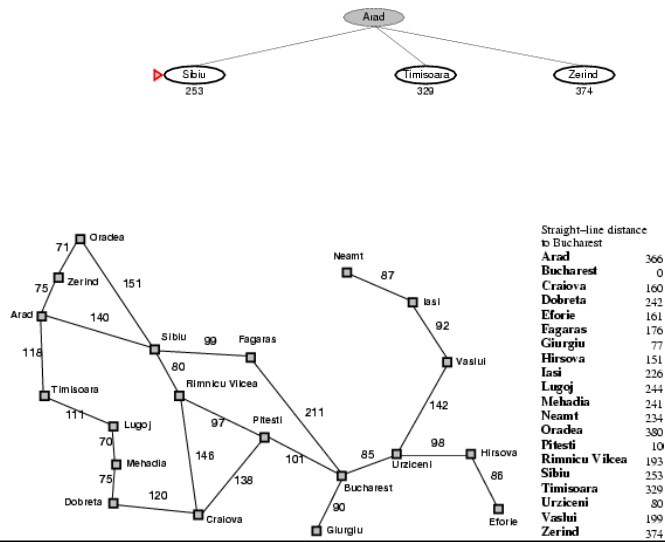
Hill Climbing search example



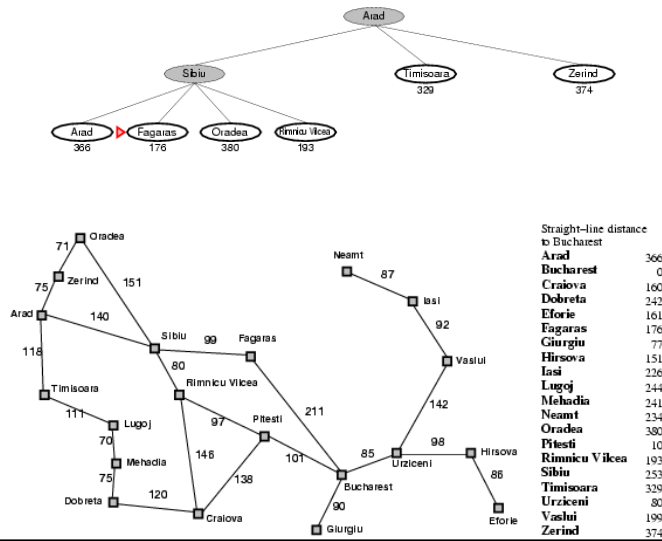
Hill Climbing search example



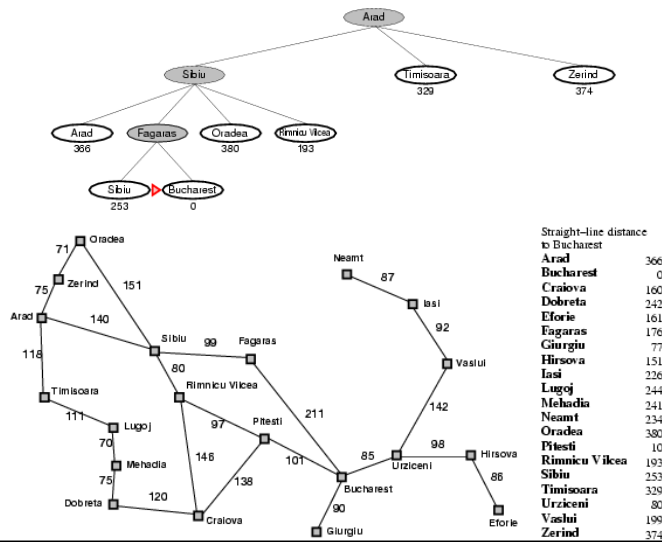
Hill Climbing search example



Hill Climbing search example



Hill Climbing search example



Hill climbing

Complete:

No, Can get stuck in loop. Complete if loops are avoided.

Time complexity?

$O(b^m)$, but with some good heuristic, it could give better results

Space complexity?

$O(b^m)$, keeps all nodes in memory

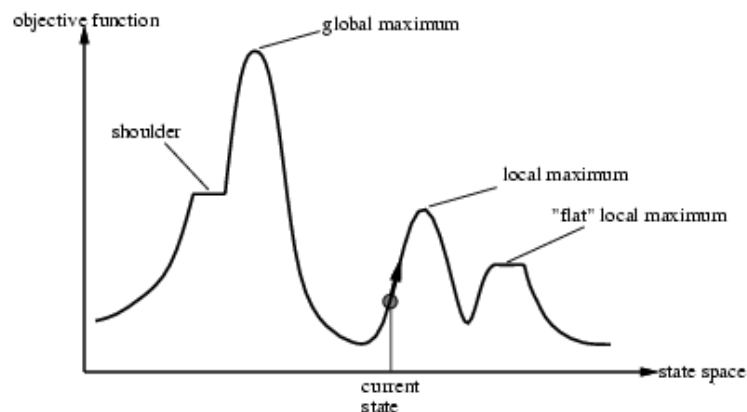
Optimality?

No

e.g. Arad → Sibiu → Rimnicu Virea → Pitesti → Bucharest is shorter!

Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima,...etc



Problems with hill climbing

1. **Local maximum** problem: there is a peak, but it is lower than the highest peak in the whole space.
2. The **plateau** problem: all local moves are equally unpromising, and all peaks seem far away.
3. The **ridge** problem: almost every move takes us down.

Solution:

Random-restart hill climbing is a series of hill-climbing searches with a randomly selected start node whenever the current search gets stuck.