Uninformed Search Strategies-2

Iterative Deepening Search (IDS)

Function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution

inputs: problem, a problem

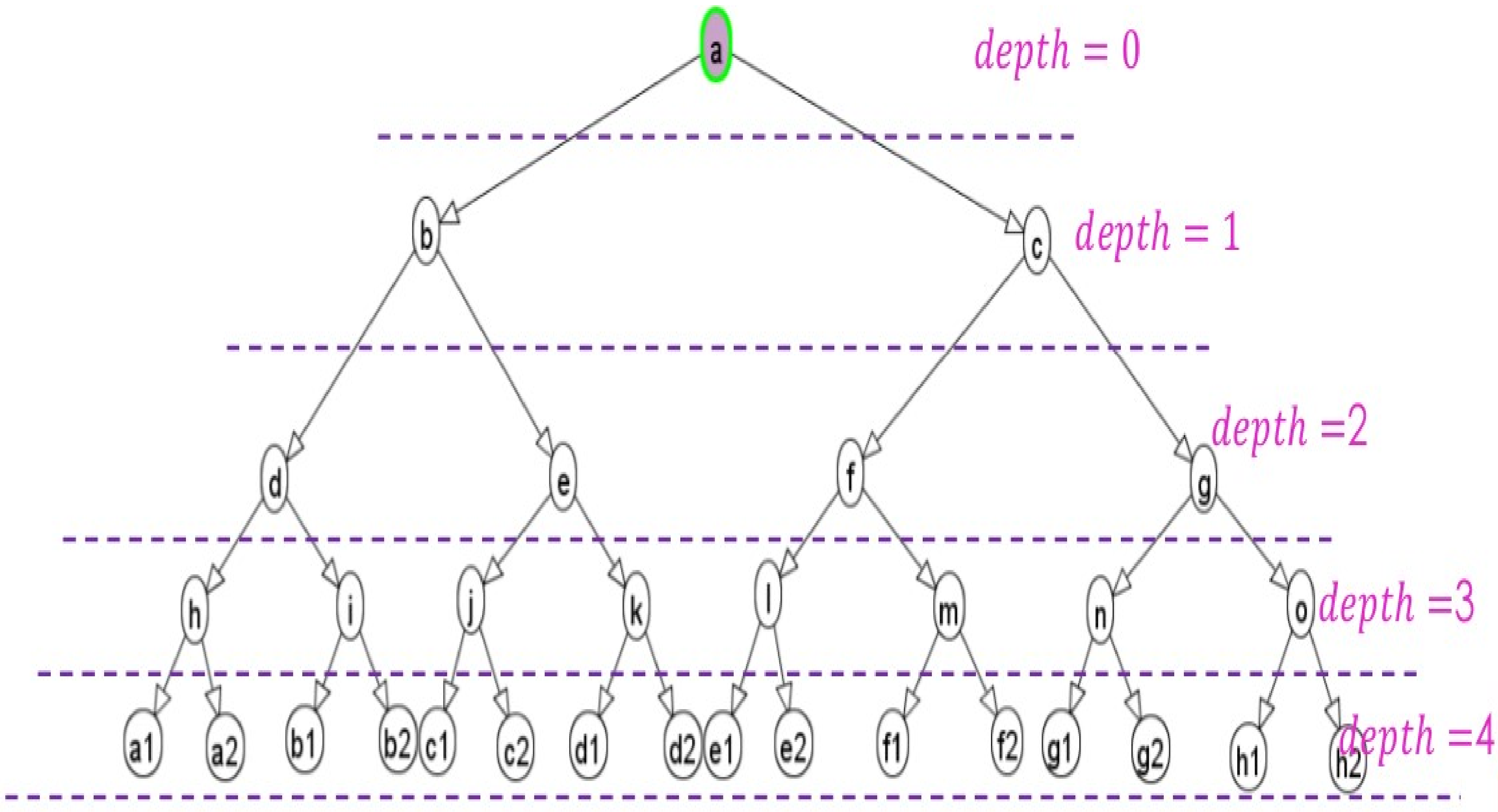
for depth = 0 to ∞ do

    result ← DEPTH-LIMITED-SEARCH(problem, depth)

    if result ≠ cutoff then return result

end

Iterative Deepening Search (IDS)

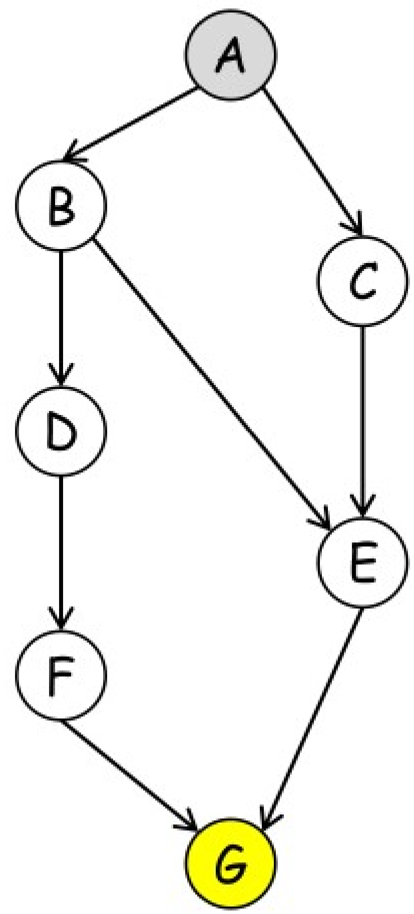


IDS Analysis

- Completeness: Yes!
- Optimality: Yes for Uniform cost edges, can be modified for non-uniform cost trees
  - Iterative lengthening search
- Time Complexity: exponential in d
- Space Complexity: bd
  - Let b=10and d=5
    - ❑$N(IDS)$=50+400+3000+20000+10000=123450
    - ❑$N(BFS)$=10+100+1000+10000+10000=111110
  - In general $N(IDS)=(d)b+(d-1)b^2+\cdots+(1)b^d \sim O(b^d)$

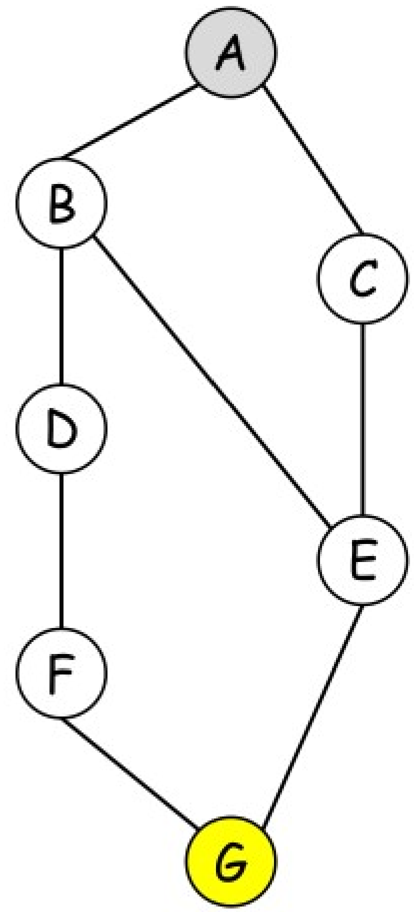
Exercise

- DFS?
- BFS?
- IDS?



Exercise

- DFS?
- BFS?
- IDS?



Problem of Redundant Paths

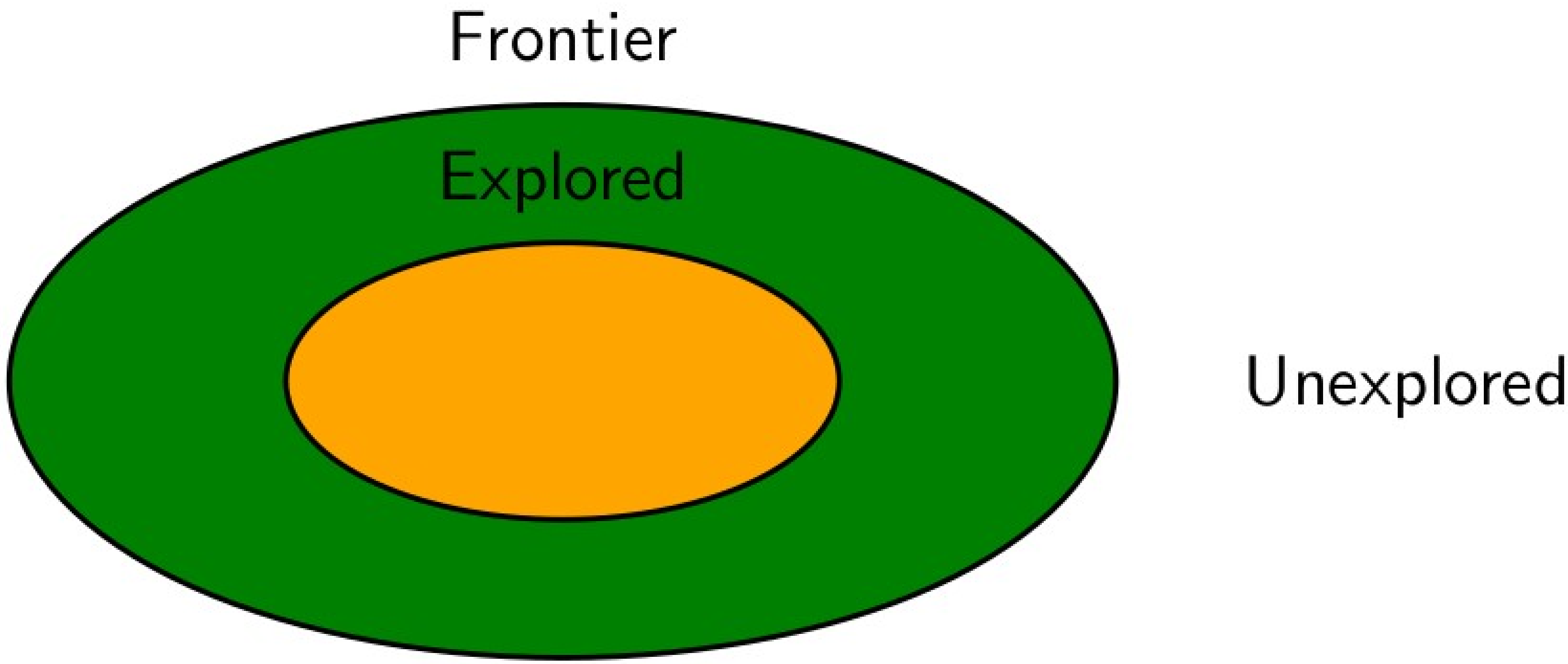
- Partial reduction in repeated expansion can be done by
  - Checking to see if any children of a node n have the same state as the parent of n
  - Checking to see if any children of a node n have the same state as any ancestor of n (at most d ancestors for n—where d is the depth of n)

Uniform Cost Search (UCS)

- Expand least-cost $g(n)$ unexpanded node
- The name uniform cost search refers to the fact that we are exploring states of the same past cost uniformly
- All action costs are non-negative
  - What if the cost associated with an edge is negative?
    - Bellman-Ford algorithm
- Implementation: Priority queue –sort the nodes in the queue based on cost

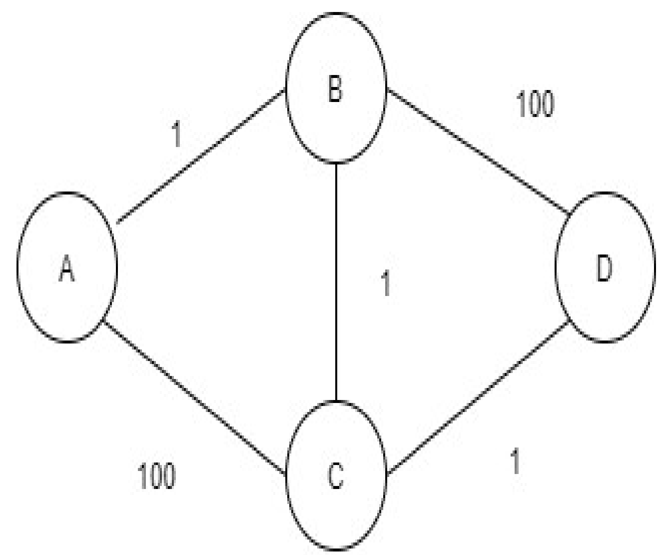
High-Level Description

- Explored: states we've found the optimal path to
- Frontier: states we've seen, still figuring out how to get there cheaply
- Unexplored: states we haven't seen



UCS Example

- Initially, we put A on the frontier. We then take A off the frontier and mark it as explored. We add B and C to the frontier with past costs 1 and 100, respectively
- Next, we remove from the frontier the state with the minimum past cost (priority), which is B. We mark B as explored and consider successors A, C, D. We ignore A since it's already explored. The past cost of C gets updated from 100 to 2. We add D to the frontier with initial past cost 101
- Next, we remove C from the frontier; its successors are A, B, D. A and B are already explored, so we only update D's past cost from 101 to 3
- Finally, we pop D off the frontier, find that it's a end state, and terminate the search



Start state : A, End state: D

Algorithm UCS

Add s_{start} to **frontier** (priority queue)

Repeat untill **frontier** is empty:

    Remove s with smallest priority p from **frontier**

    If IsEnd(s): return solution

    Add s to **explored**

    For each action a ∈ Actions(s):

    Get successor s’ ← Succ(s, a)

    If s’ already in **explored**: continue

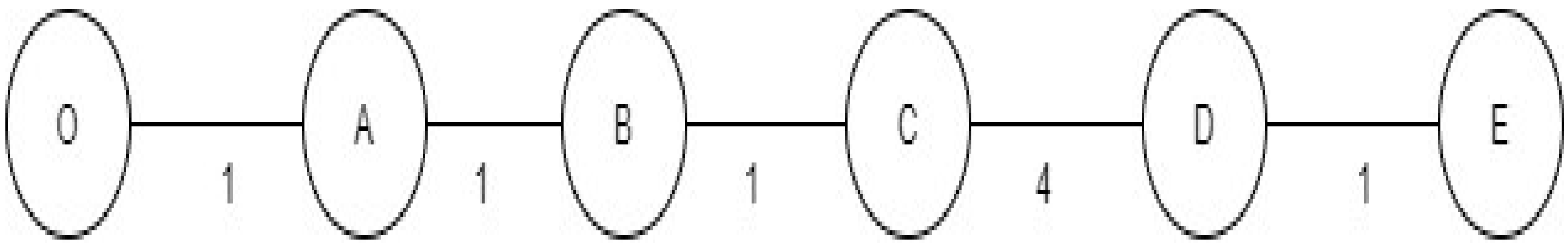
    Update **frontier** with s’ and priority p + Cost(s, a)

Much like Dijkstra’s Shortest-path Algorithm

- UCS takes as input a search problem, which implicitly defines a large and even infinite graph, whereas Dijkstra’s algorithm (in the typical exposition) takes as input a fully concrete graph
- UCS finds the shortest path to the goal state

Issue with UCS

- May search for useless inexpensive large subtrees before useful path with costly steps



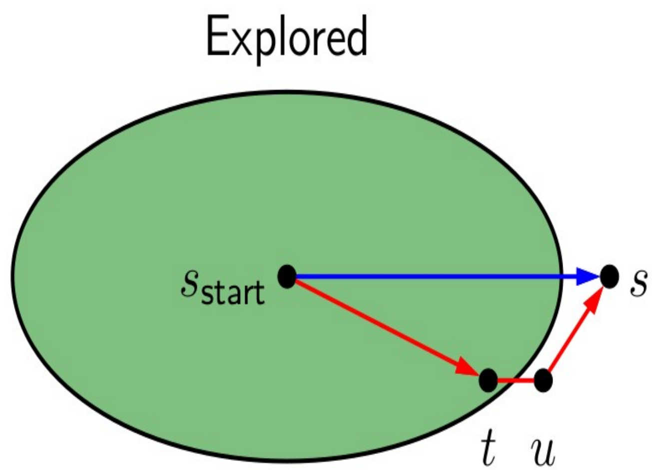
Start state : C, End/Goal state: E

Analysis of UCS: Correctness Theorem

- ❑ When a state s is popped from frontier and moved to explored, its priority is PastCost(s), the minimum cost to s

<u>Proof:</u>

- Let p_s be the priority of s when s is popped off the frontier. Since all costs are non-negative, p_s increases over the course of the algorithm
- Suppose we pop s off the frontier. Let the blue path denote the path with cost p_s
- Consider any alternative red path from the start state to s. The red path must leave the explored region at some point; let t and u = Succ(t, a) be first pair of states straddling the boundary. We want to show that the red path can't be cheaper than the blue path via a string of inequalities



Analysis of UCS: Correctness Theorem

- First, by definition of PastCost(t) and non-negativity of edge costs, the cost of the red path is at least the cost of the part leading to u, which is PastCost(t) + Cost(t, a) = p_t + Cost(t, a)
- Second, we have p_t + Cost(t, a) $\geq p_u$ since we updated the frontier based on (t, a)
- Third, we have that $p_u \geq p_s$ because s was present at the top of the frontier
- Note that p_s is the cost of the blue path

UCS Analysis

- Completeness: Yes; if step cost $\geq \epsilon$
- Optimality: Yes; nodes are expanded in increasing order of $g(n)$
- Time Complexity: # of nodes with $g \leq$ cost of optimal solution - $O(b^{\lceil C^*/\epsilon \rceil})$
- Space Complexity: # of nodes with $g \leq$ cost of optimal solution - $O(b^{\lceil C^*/\epsilon \rceil})$

Exercises

- Can UCS handle cyclic graphs?
- UCS can only deal with non-negative action costs. As an alternative solution one may add a large positive constant to each action cost to make them all non-negative, and subsequently solve the problem. Will this strategy lead to correct solution? Why or why not?

Performance of Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

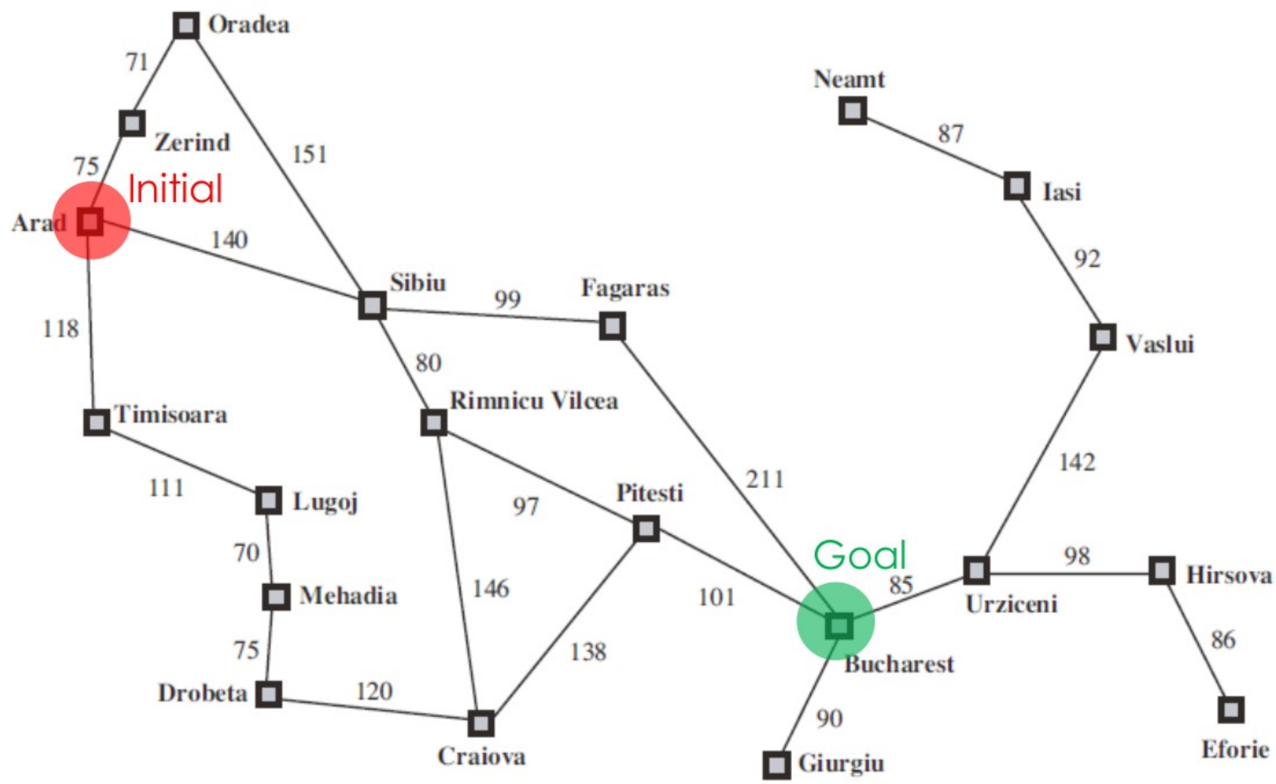
Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Hints on Previous Exercises

- In the game of chess which strategy you would like to prefer to find the winning state?
- In Tic-Tac-Toe, define the states, initial-state, actions, goal test and path cost.
- Whether the tree associated with the Tic-Tac-Toe will be a search tree?
- Tic-Tac-Toe, which strategy should be adopted to avoid loosing the game?
- Whether the search tree of Tic-Tac-Toe will be having the redundancy in terms of paths or nodes? Justify your answer.

Next Class

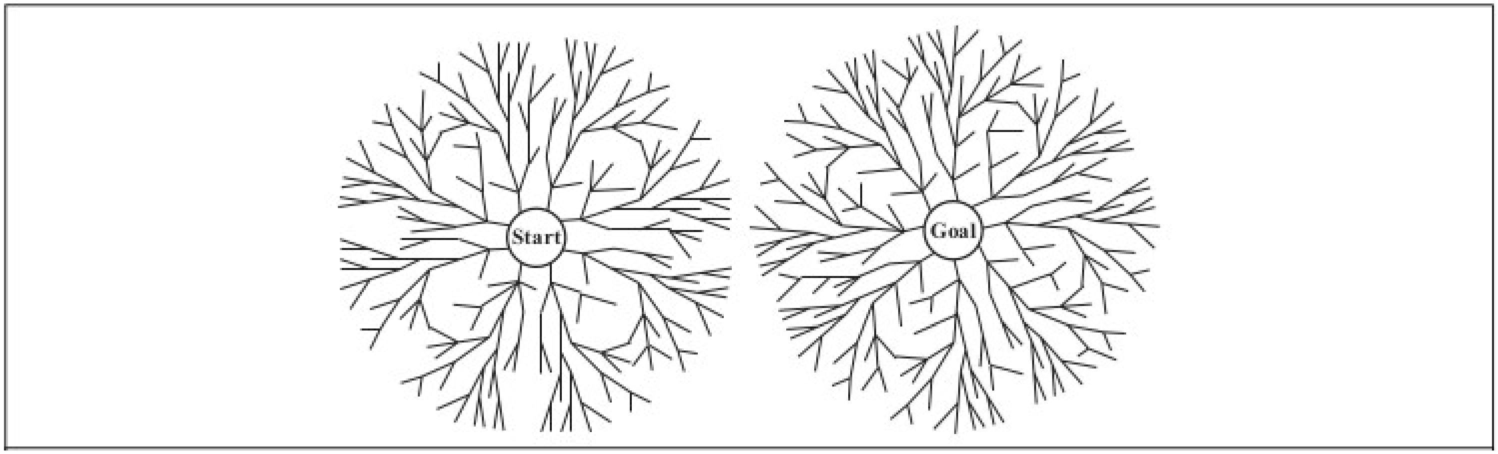
- Bidirectional Search
- Informed Search strategies:
  - Greedy best first search
  - A* search



References

- CS188 UCB course
- Professor Mausam IIT Delhi AI course
- Rusell and Norvig text book

Bi-directional Search



- Run two simultaneous searches
  - Forward search from the initial state
  - Backward search from the goal state
- Replace the goal test function with a check to see whether the frontiers of the two searches intersect

Bi-directional Search Analysis

- Complete: Yes (b should be finite)
- Time complexity: O($b^{d/2}$)
- Space complexity: O($b^{d/2}$)
- Optimal: Yes (if uniform cost search is used in both directions along with a hash table)
- Single goal state vs multiple goal states
- Time and space issues

Exercises

- Go through and analyse the iterative analogue of UCS i.e. iterative lengthening search. Compare it with UCS on 8-puzzle problem
- Describe a state space in which iterative deepening performs much worse than depth-first search
- The heuristic path algorithm is a best first-search in which the evaluation function is $f(n)=(2-w)*g(n) + w*h(n)$. For what values of w is this complete?