Artificial Intelligence

Uninformed search techniques

Dr. Priyadarshan Dhabe,
Ph.D-IIT Bombay,
Professor in Information Technology

Syllabus-

- **Uninformed Search strategies**

Formulation of real world problems, Breadth first search, Depth first search, Depth limited search, Iterative deepening depth first search, bidirectional search, Comparison of uninformed search strategies, searching with partial information, sensor less problems, Contingency problems

Credits:- Algorithms and figures are taken from the book of Russell and Norvig, " artificial intelligence -a modern approach"

AI Problem Solving Techniques

Searching is the important technique to solve many AI problems

1. **Uninformed Search/Blind Search**
2. **Informed /Heuristic search**

What are uninformed search strategies?

-They are also called as "**blind search**" techniques

-**Definition**:- Search technique which "**do not have additional information**" about states beyond that provided in problem definition.
-They can only generate successors and recognize the goal state and non-goal state.

Features of uninformed/blind search

- Searches the state space in an exhaustive way
- Can distinguish between goal and non-goal state
- They don't have **additional info** to guide the search.

Why we need uninformed search strategies?

We do not have additional information (clue/hint) needed to quickly solve the problem

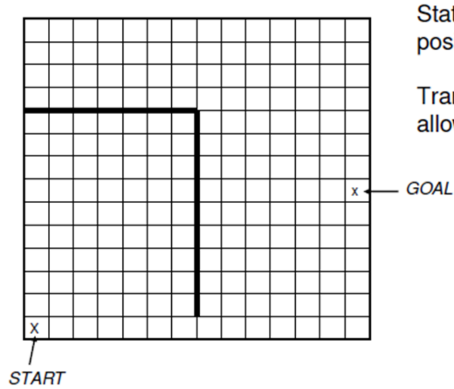
(Source: wikipedia)

Missing Malaysian flight MH370

-**Example1:**- Searching a location in the city assuming no body is present to help you
-**Example2:**- Finding location of interest on Mars.

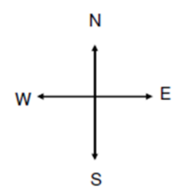
Formulation of search problem-Robot navigation

Example: Robot Navigation



States = positions in the map

Transitions = allowed motions



Navigation: Going from point START to point GOAL given a (deterministic) map

http://www.cs.cmu.edu/~awm/tutorials

Real world (generic)problem formulation

Formulation

- Q: Finite set of states
- $S \subseteq Q$: Non-empty set of start states
- $G \subseteq Q$: Non-empty set of goal states
- **succs**: function $Q \rightarrow P(Q)$
  succs(s) = Set of states that can be reached from s in one step
- **cost**: function $Q \mathrm{x} Q \rightarrow$ Positive Numbers
  cost(s,s') = Cost of taking a one-step transition from state s to state s'
- Problem: Find a sequence $\{s_1,\ldots,s_K\}$ such that:

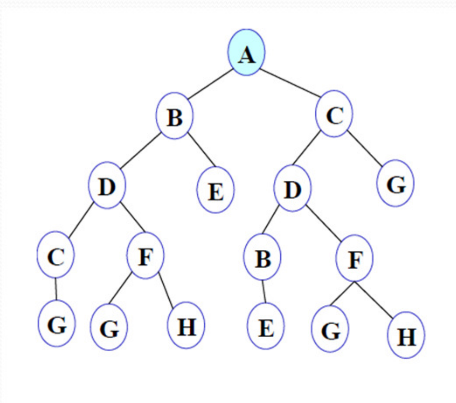
1. $s_1 \in S$
2. $s_K \in G$
3. $s_{i+1} \in$ **succs**(s_i)
4. $\sum$ **cost**(s_i, s_{i+1}) is the smallest among all possible sequences (desirable but optional)

http://www.cs.cmu.edu/~awm/tutorials

Breadth first search

- The root node is expanded first, then all the successors and so on.
- In general all the nodes are expanded at a given depth in the search tree before any nodes at the next level
- BFS uses TREE-SEARCH algorithm with an empty fringe, that is a FIFO queue, assuring that all the nodes visited first will be expanded first.
- Newly generated successor nodes are kept at the end of queue and hence shallow nodes are expanded before deeper nodes.

Apply Breadth first search- FIFO fringe



Let G is a goal node

Fringe- is a FIFO queue

A initially

B,C

C,D,E

D,E,D,G

E,D,G, C, F

D,G, C, F E has no child

G, C, F,B,F

G is Goal node, thus algo. halts

Breadth first search-Analysis

- BFS has exponential time and space complexity.
- Let
  - b- be the branching factor of non-leaf nodes
  - d- be the depth of search tree
- Then total nodes on search tree

$$1+b+b^2+....,+b^d = (b^{(d+1)}-1)/(b-1)$$

- Thus BFS has time and space complexity of $O(b^d)$

Breadth first search-Analysis

Let d=15 and b=10, then BFS will generate 10^{15} nodes and thus requires lot of time. Even if we have time it will run out of memory soon.

So BFS is efficient for small search spaces only.

Advantages of Breadth First Search

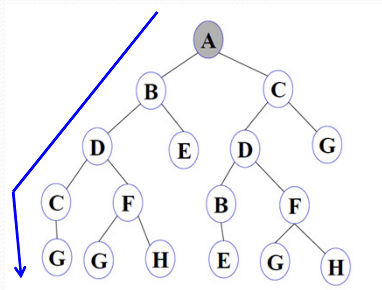
Finds the path of minimal length to the goal.

Disadvantages of Breadth First Search

Requires the generation and storage of a tree whose size is exponential the the depth of the shallowest goal node

Depth first search- LIFO fringe

- The fringe is LIFO now



A initially

B,C

D,E,C

C,F,E,C

G,F,E,C

G is Goal node, thus algo. halts

Depth first search-Analysis

DFS takes worst case exponential time i.e $O(b^d)$.

Let N be the maximum depth of a node in search tree.

However the space taken is linear in the depth of search tree i.e O(bN)

If search tree has infinite depth DFS may not terminate

Justification : - space needed at each level = b and maimum depth = N

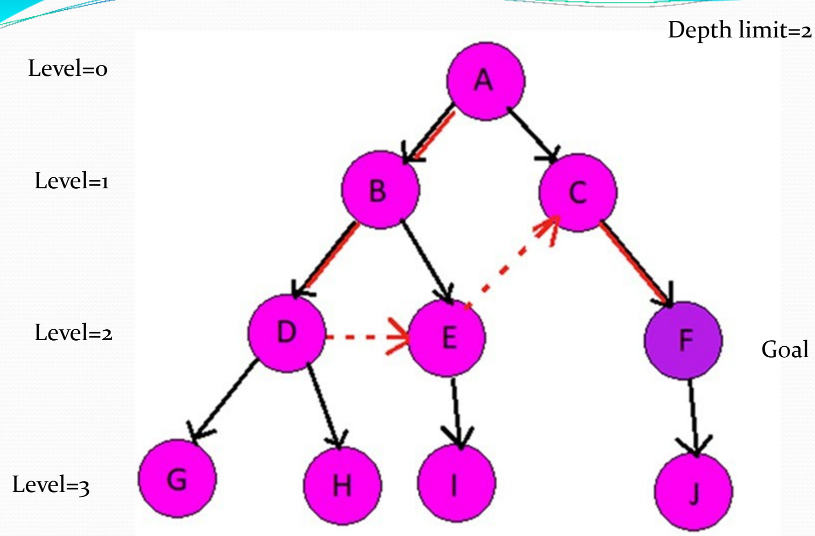
Thus, the space complexity = O(bN)

If the search tree has infinite depth (due to cycles) then algorithm may not terminate, this drawback can be eliminated using Depth limited search Algorithm

Depth Limited Search

- It is a variant of DFS that uses a depth "Bound/limit".
- Nodes are only expanded if they have depth less than or equal to the bound. Fringe is LIFO structure.

```
Depth limited search (limit)
Let fringe be a list containing the initial state
Loop
        if fringe is empty return failure
        Node ← remove-first (fringe)
         if Node is a goal
             then return the path from initial state to Node
        else if depth of Node = limit return cutoff
        else add generated nodes to the front of fringe
End Loop
```



https://www.youtube.com/watch?v=P7WQUBLKDmo

Depth Limited Search-Analysis

Let l be the depth limit and d be the actual depth of search tree

DLS may not find solutions present at depths $> l$

DLS will also be nonoptimal if we choose $l > d$

DLS has time and space complexity of $O(b^l)$ and $O(bl)$

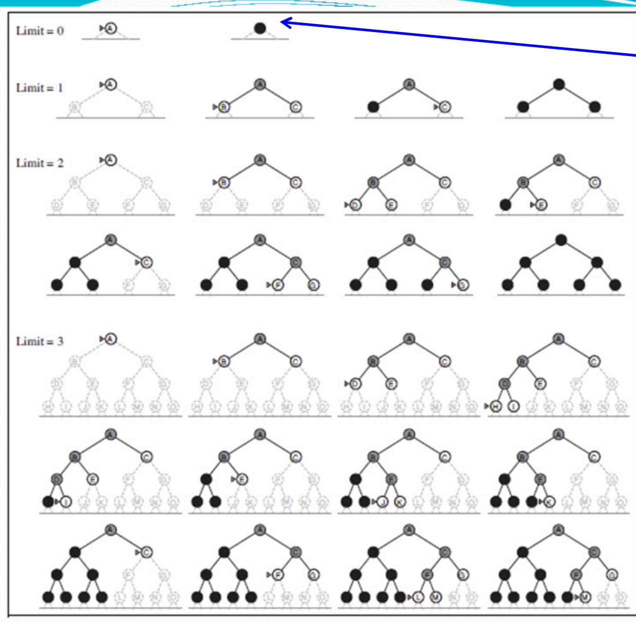
DLS **can report two kinds of failure**

standard failure and cut-off

Iterative Deepening Search

- It is also called Iterative deepening Depth First Search
- It combines benefits of DFS and BFS.
- Per iteration it increases the depth limit by 1.

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
   for depth = 0 to ∞ do
      result ← DEPTH-LIMITED-SEARCH(problem, depth)
      if result ≠ cutoff then return result
```



Root is at level 0

Four iterations of iterative deepening search

Iterative Deepening Search (IDS)

- The upper levels of search tress are generated multiple times and thus, IDS needs more efforts.
- But it is not very costly, since most of the nodes are present at the bottom of the tree.

$$N = (d)b + (d-1)b^2 + ... + (1)b^d$$

-IDS is preferred blind search method when there is large search space and depth of solution is not known.
-It finds shallowest solution (solution at lower levels)

Iterative Deepening Search (IDS)

In IDS nodes at the bottom level (depth d) are generated once, thus total nodes generated (N) in IDS is

$$N = (d)b + (d-1)b^2 + \ldots.. + (1)b^d$$

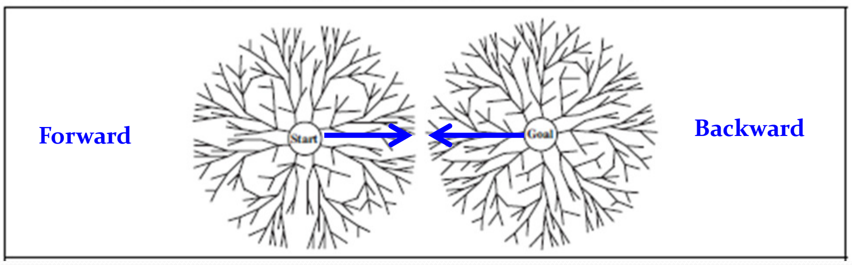
b^d are created only once, then b^(d-1) nodes are created twice and so on.... and b nodes are created d times

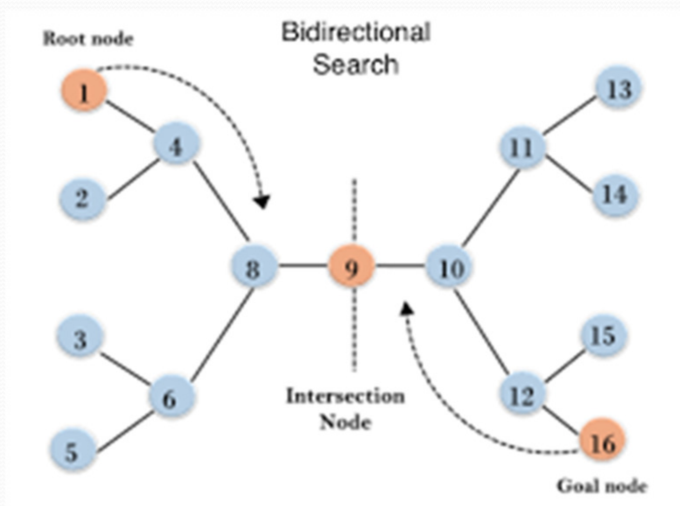
Iterative Deepening Search (IDS)

- Class exercise 1:-
- Compare BFS and IDS for some combinations of d and b. (refer :- Russell and Norvig)

Bidirectional search

- It uses two simultaneous searches one forward –from initial state and the other backward from the goal, and stops when two searches meet in the middle.
- Each search takes alternate turns.





Bidirectional search

- If a problem has solution depth=6, and each direction runs BFS one node at a time, then in worst case two searches will meet at depth=3.
- Both searches check each node before it is expanded, to see if it is the fringe of other search. If so, a solution has been found.

Time and space complexity of BDS is $O(b^{d/2})$

Justification : -If both search uses breadth FS,

$$\text{total work} = O(2b^{d/2}) \Rightarrow O(b^{d/2})$$

Bidirectional search

- Reduction in time complexity makes BDS attractive.
- But how to search backward? Is a problem.
  - Using two functions Predecessor and Successor
  - *Pred(n)*- gives all the predecessors of node n
  - *Succ(n)*- gives all the successors of node n
- Forward search will use *Succ( )* and backward will use *Pred( ).*
- For BDS is good to have <u>one or very few goal states</u> e.g in 8-puzzle there is only one goal state that makes it easy to search backward.
- More goal states (like in chess) makes backward search difficult.

Comparison of uninformed search strategies

Criterion	Breadth First Search	Depth First search	Depth limited	Iterative deepening	Bidriectional
Complete?	YES	No	No	Yes if b is finite	Yes if both directions uses BFS (Y)
Time	$O(b^{d+1})$	$O(b^{m})$	$O(b^{l})$	$O(b^{d})$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(bd)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	YES	No	No	Yes, if all steps has same cost (X)	Yes, if X and Y

$m-$is maximum depth of any branch, $l-$ is depth limit

$d-$ is actual depth of tree, $b-$ is branching factor

- Read about searching with partial information

Searching with partial information

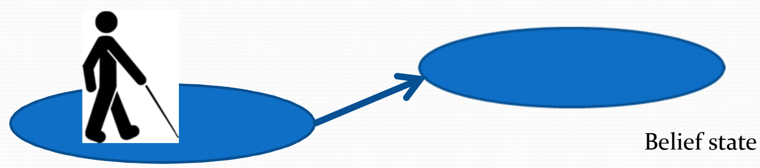
- Most of the time, an agent works in a fully observable and deterministic environment.
- Thus, agent can compute exactly the state resulting from applying a sequence of actions.
- What happens when knowledge of states or actions is incomplete?
- This in-completeness leads to 3 kinds of problems
    1. **Sensor less problems**
    2. **Contingency Problems**
    3. **Exploration problems**

Searching with partial information

- **Sensor less problems:-** (conformant Problems) If an agent has no sensors at all, then it could be in one of the several initial state and each action might lead to one of the several successor states.
- E.g (movement of visually impaired person)

Sensor less problems

- When the world is not fully observable, the agent must reason about *sets of states* it might get to , rather than single states. Such sets of states is called "***belief state***", representing belief about possible physical states.
- Exercise2:- Learn vacuum cleaner agent example from Russell and Norvig



Searching with partial information

- **Sensor less problems**
- How to solve sensor less problems?
- We search in the space of belief states rather than physical states.
- The initial state is a belief state and each action maps from one belief state to another.
- A path now connects several belief states and a solution is now a path that leads to a belief state, <u>all of whose members are goal states.</u>

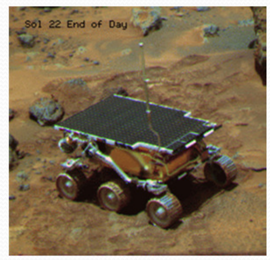
Searching with partial information

- **Contingency Problems:-**

  (Contingency is unpredicted event)

- Agent has sensors to sense its environment and can get new information (called Contingency).
- No "fixed action sequence" guarantees a solution and hence agent can not plan.

NASA path finder Mars exploration



Searching with partial information

Contingency Problems:-

-The solution to the contingency problem often takes the form of a tree, where each branch may be selected based on percept received at that point in the tree.

- Start acting and see Which contingencies do arises?

Exercise3:- Learn working of an agent in Murphy's Law World from Russell and Norvig

Searching with partial information

- **Exploration Problems:-**
- When the states and actions of the environment are unknown, the agent must act to discover them.
- These are extreme cases of contingency problems.
- Agents discover states
- Agents optimizes a function to decide best possible action