COMP111: Artificial Intelligence

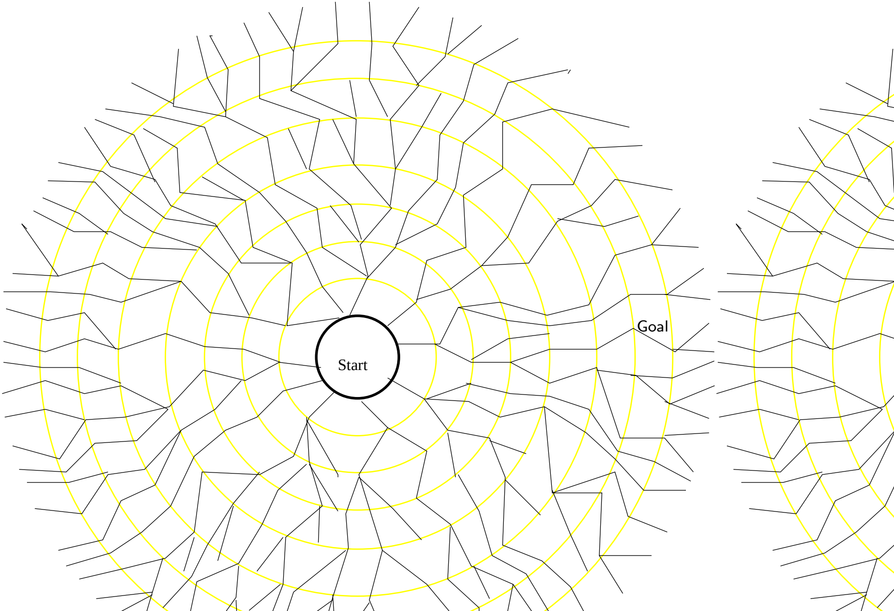
Section 5. More on uninformed tree search

Frank Wolter

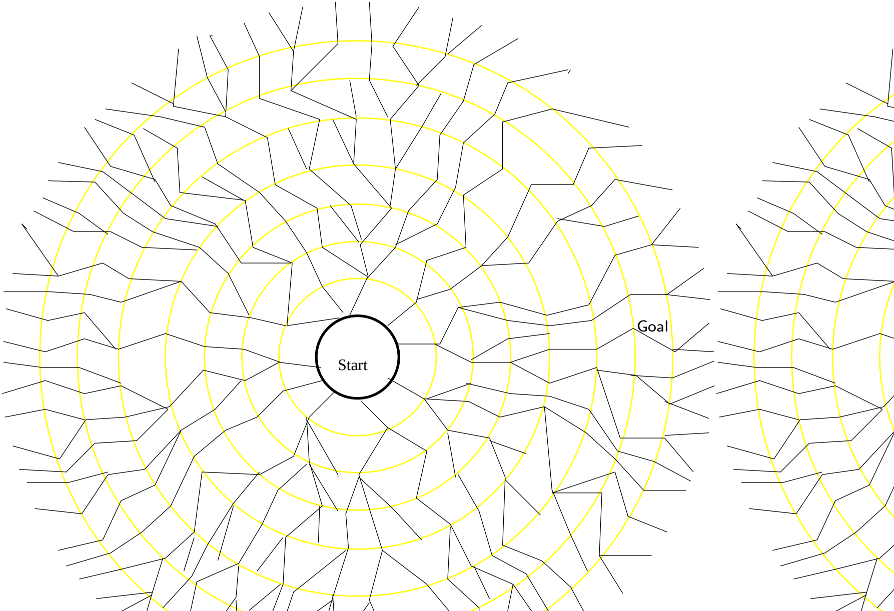
Recap

- Basic uninformed search tree algorithms:
  - **Breadth-first search**
    complete but expensive.
  - **Depth-first search**
    desirable space complexity but incomplete

Example: BFS in a Maze



Example: DFS in a Maze



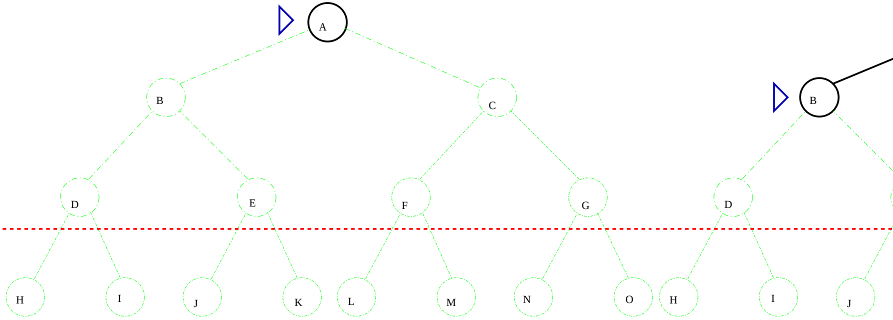
Overview

- Variations and combinations:
  - Limited depth search (LDF)
  - Iterative deepening search (IDF)
- Speeding up techniques
  - Avoiding repetitive states
  - Bi-directional search

Depth Limited Search (DLS)

- Depth-first search has some desirable properties — space complexity.
- But if wrong path expanded (with no solution on it), then search may take very long or it may even not terminate.
- Idea: introduce a depth limit on paths to be expanded.
- Don't expand a path that is longer.
- Useful if you know the maximum length of the solution.

Depth Limited Search (II)

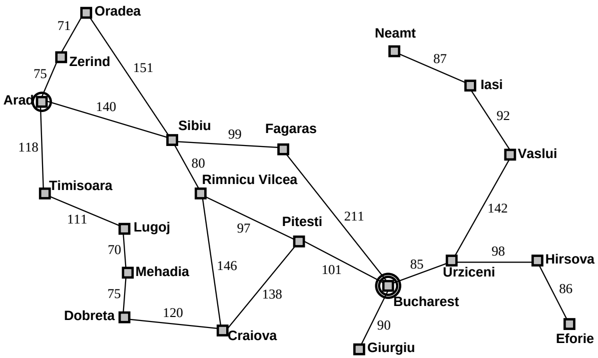


Depth Limited Search (DLS)

Input: a start state s_0
for each state s the successors of s
a test goal(s) checking whether s is a goal state
non-negative integer DepthLimit

Set frontier $:= \{s_0\}$
while frontier is not empty **do**
    select and remove from frontier the path $s_0 \ldots s_k$ added
    last to frontier
    if goal(s_k) **then**
        return $s_0 \ldots s_k$ (and terminate)
    else If $k <$ DepthLimit, then for every successor s of s_k
        add $s_0 \ldots s_k s$ to frontier
    end if
end while

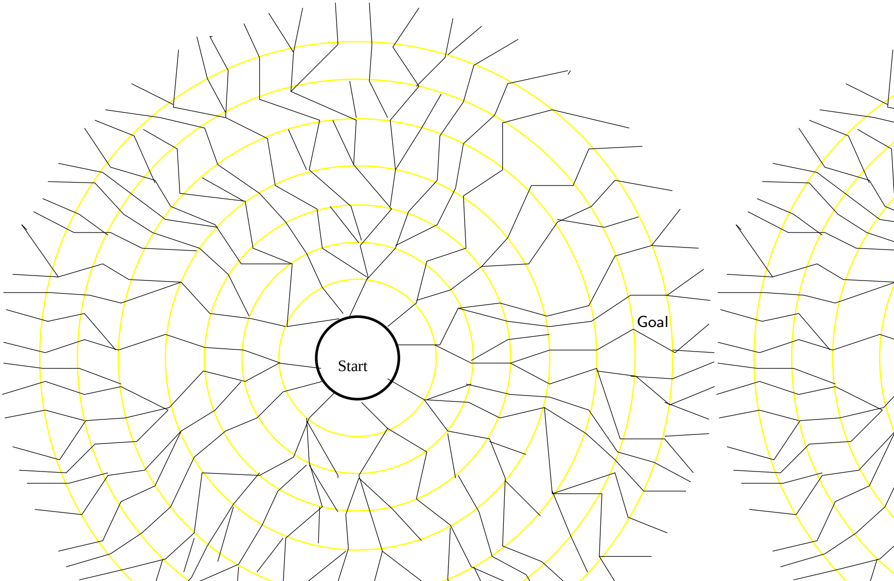
Example: Romania Problem



- Only 20 cities on the map $\Longrightarrow$ no path without repeating states longer than 19.
- Careful examination shows that any city can be reached from any other city in less than 10 steps. So we could choose a depth limit of 10.

Example: DLS in a Maze

Level = 2



Properties of Depth Limited Search (DLS)

- Not complete: if the length of the shortest path to a goal state is longer than the depth limit then DLS will not find it. Otherwise it will.
- Not optimal: Solution found is not guaranteed to be a shortest path.
- Time complexity: let l be the depth limit. Then in the worst case DLS will look at

$$1 + b + b^2 + \cdots + b^l$$

  paths before reaching a goal state.
- Space complexity: let l be the depth limit. Then in the worst case the frontier can contain bl paths.

Iterative Deepening

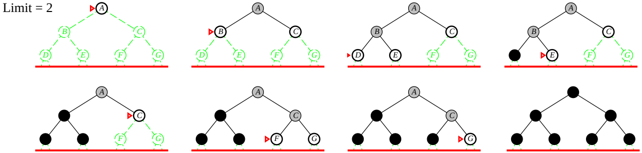
- Unfortunately, if we choose a maximal depth for DLS such that shortest path is longer, then DLS is not complete.
- Iterative deepening is a complete version of it.
- Basic idea is:
  - do DLS for depth $n = 0$; if solution found, return it;
  - otherwise do DLS for depth $n = n + 1$; if solution found, return it, etc;
- So we repeat DLS for all depths until solution found.
- Useful if the search space is large and the maximum depth of the solution is not known.

Iterative deepening search $l = 0$

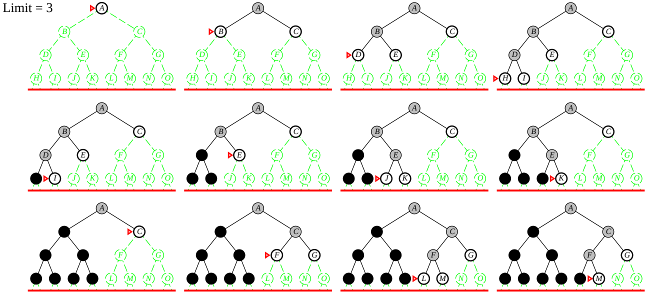
Limit = 0

Iterative deepening search $l = 1$

Iterative deepening search $l = 2$



Iterative deepening search $l = 3$



General Algorithm for Iterative Deepening (IDS)

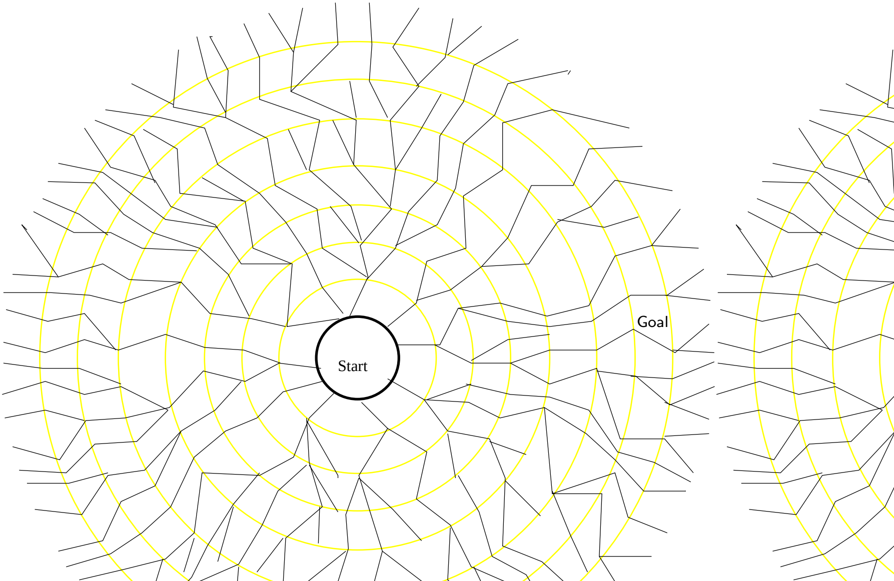
Call DLS as a subroutine:

Input: a start state s_0
    for each state s the successors of s
    a test goal(s) checking whether s is a goal state

Set DepthLimit := 0
repeat
    Result := depth_limited_search(DepthLimit)
    if Result is a path $s_0 \ldots s_k$ **then**
        return $s_0 \ldots s_k$ (and terminate)
    else DepthLimit := DepthLimit + 1
    end if
until False

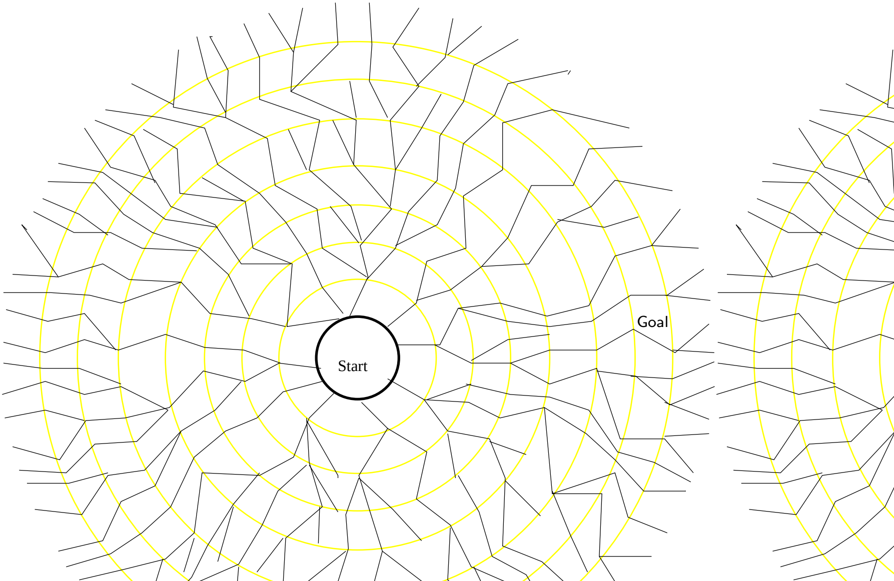
Example: IDS in a Maze

Level = 1



Example: IDS in a Maze

Level = 2



Properties of Iterative Deepening (IDS)

- Complete: IDS always finds a path to a goal state if there exists a path.
- Optimal: Solution found is guaranteed to be a shortest path.
- Time complexity: let d be the length of the shortest path to a goal state. Then in the worst case IDS will look at

  $$1 + (1 + b) + (1 + b + b^2) + \cdots + (1 + b + b^2 + \cdots + b^d)$$

  paths before reaching a goal state.
- Space complexity: let d be the length of the shortest path to a goal state. Then in the worst case the frontier can contain bd paths.

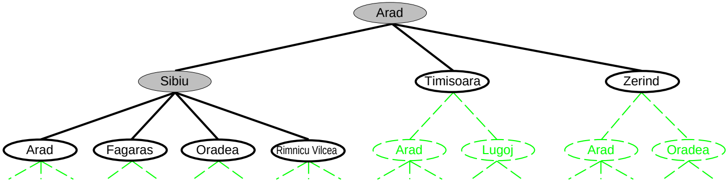
IDS versus BFS

- Note that in iterative deepening, we regenerate paths in each iteration: each time we do call the DLS algorithm for depth d, we need to regenerate the tree to depth $d - 1$.
- Comparison with BFS:
  - Trade off time for memory.
  - In general we might take a little more time, but we save a lot of memory.
  - Example: Suppose $b = 10$ and $d = 5$. Breadth-first search would require examining $1 + b + \cdots + b^d = 111,111$ paths, with memory requirement of $b^d = 100,000$ paths.
    Iterative deepening for the same problem:
    $1 + (1 + b) + (1 + b + b^2) + \cdots + (1 + \cdots + b^d) = 123,456$ paths to be searched, with memory requirement only $bd = 50$ paths.
    Takes 11% longer in this case.

Algorithmic Improvements

Repeated States

From the Romania example you have probably noticed the generation of states that have previously been explored.



Doing DFS we can create an infinite path!

Avoiding Repeated States

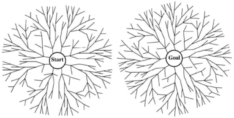
There are three ways to deal with this (in order of increasing effectiveness and computational overhead):

- do not return to the state you have just come from;
- do not create paths with cycles in them (two occurrences of the same state);
- do not create paths containing states that occurred in a path created earlier.

Note there is a trade off between the cost of extra search and the cost of checking for repeated states.

Bi-directional Search

- Suppose we search from the goal state backwards as well as from initial state forwards.
- Involves determining predecessor states to goal, and then looking at predecessor states to this, . . .



Bi-directional Search: good properties

- Often much more efficient.
- Rather than doing one search up to depth d, we do two searches up to depth $d/2$.
- Example:
  Suppose $b = 10$, $d = 6$.
  Breadth-first search will examine
  $1 + b + \cdots + b^6 = 1,111,111$ paths.
  Bidirectional search will examine
  $2 \times (1 + b + b^2 + b^3) = 2,222$ paths.
- Can combine different search strategies in different directions.

Bi-directional Search: problematic properties

- Must be able to generate predecessors of states.
- There must be an efficient way to check whether a new state appears in the other search.
- For large d, still impractical!
- For two bi-directional breadth-first searches, with branching factor b and depth of the solution d we have memory requirement of $b^{d/2}$ for each search.

Summary

- More advanced problem solving techniques:
  - Depth limited search
  - Iterative deepening
  - Bi-directional search
  - Avoiding repeated states
- These often improve on basic techniques like breadth-first and depth-first search.
- However, still mostly they are not powerful enough to give solutions for realistic problems.