Basic to problem solving:

• How to take action to reach a goal?
Choices have consequences!
Search

Formalizing the problem statement …

• Problem can be in various states.
• Start in an *initial state*.
• Have some *actions* available.
• Each action has a cost.
• Want to reach some *goal*, minimizing cost.

Happens in simulation.

*Not* web search.
Formal Definition

Set of states $S$

Start state $s \in S$

Set of actions $A$ and action rules $a(s) \rightarrow s'$

Goal test $g(s) \rightarrow \{0, 1\}$

Cost function $C(s, a, s') \rightarrow \mathbb{R}^+$

So a search problem is specified by a tuple, $(S, s, A, g, C)$. 
Problem Statement

Find a sequence of actions $a_1, \ldots, a_n$ and corresponding states $s_1, \ldots, s_n$

... such that:

\begin{align*}
  s_0 &= s \\
  s_i &= a_i(s_{i-1}), \quad i = 1, \ldots, n \\
  g(s_n) &= 1
\end{align*}

start state

legal moves

end at the goal

while minimizing:

$$
\sum_{i=1}^{n} C(s_{i-1}, a, s_i) \quad \text{minimize sum of costs - rational agent}
$$
Formal Models

What are they good for?
Example

Sudoku

States: all legal Sudoku boards.

Start state: a particular, partially filled-in, board.

Actions: inserting a valid number into the board.

Goal test: all cells filled and no collisions.

Cost function: 1 per move.
Example

*Flights* - e.g., *ITA Software*.

States: airports, times.

Start state: TF Green, 5pm.

Actions: available flights from each airport.

Goal test: reached Tokyo by midnight tomorrow.

Cost function: time and/or money.
The Search Tree

Classical conceptualization of search.
The Search Tree
Important Quantities

Breadth (branching factor)
The Search Tree

Depth
- min solution depth \( m \)
- depth \( d \)

\[ O\left(b^d\right) \] leaves in a tree of breadth \( b \), depth \( d \).

\[ \sum_{i=0}^{d} b^i \in O\left(b^d\right) \] total nodes in the same tree
The Search Tree

Expand the tree one node at a time.
Frontier: set of nodes in tree, but not expanded.

Key to a search algorithm: which node to expand next?
visited = {}
frontier = {s0}
goal_found = false

while not goal_found:
    node = frontier.next()
    frontier.del(node)
    if(g(node)):
        goal_found = true
    else:
        visited.add(node)
        for child in node.children:
            if(not visited.contains(child)):
                frontier.add(child)
How to Expand?

*Uninformed strategy:*
- nothing known about likely solutions in the tree.

What to do?
- Expand deepest node (*depth-first search*)
- Expand closest node (*breadth-first search*)

Properties
- Completeness
- Optimality
- Time Complexity (*total number of nodes visited*)
- Space Complexity (*size of frontier*)
Depth-First Search

Expand deepest node

s0

s1
Depth-First Search

Expand deepest node
Depth-First Search

Expand deepest node
Depth-First Search

Expand deepest node
Depth-First Search

Expand deepest node
DFS: Time

worst case: solution on this branch

\[ O(b^d - b^{d-m}) = O(b^d) \]
DFS: Space

worst case: search reaches bottom

\[ O((b - 1)d) = O(bd) \]
Depth-First Search

Properties:
• Completeness: Only for finite trees.
• Optimality: No.
• Time Complexity: $O(b^d)$
• Space Complexity: $O(bd)$

Note that when reasoning about DFS, $m$ is depth of found solution (not necessarily min solution depth).

The deepest node happens to be the one you most recently visited - easy to implement recursively OR manage frontier using LIFO queue.
Breadth-First Search

Expand shallowest node
Breadth-First Search

Expand shallowest node

\[ s_0 \]

\[ s_1 \]

\[ s_2 \]
Breadth-First Search

Expand shallowest node
Breadth-First Search

Expand shallowest node
Breadth-First Search

Expand shallowest node
BFS: Time

\[ O(b^{m+1}) \]
BFS: Space

\[ O(b^{m+1}) \]
Breadth-First Search

Properties:
• Completeness: Yes.
• Optimality: Yes for constant cost.
• Time Complexity: $O(b^{m+1})$
• Space Complexity: $O(b^{m+1})$

Better than depth-first search in all respects except memory cost - must maintain a large frontier.

Manage frontier using FIFO queue.
Iterative Deepening Search

Combine these two strengths.

The core problems in DFS are a) *not optimal*, and b) *not complete* … because it fails to explore other branches.

Otherwise it’s a very nice algorithm!

Iterative Deepening:
• Run DFS to a fixed depth $z$.
• Start at $z=1$. If no solution, increment $z$ and rerun.
IDS

run DFS to this depth
IDS

Optimal for constant cost! Proof?

How can that be a good idea?

*It duplicates work.*

Sure but:

- Low memory requirement (equal to DFS).
- Not many more nodes expanded than BFS. (About twice as many for binary tree.)
IDS

visited $m + l$ times

visited $m$ times

...
IDS (Reprise)

\[
\sum_{i=0}^{m} b^i (m - i + 1) = \frac{b(b^{m+1} - m - 2) + m + 1}{(b - 1)^2}
\]

# nodes at level \(i\)

# revisits

DFS worst case: \[
\frac{b^{m+1} - 1}{b - 1}
\]
IDS

Key Insight:
• Many more nodes at depth $m+1$ than at depth $m$.

MAGIC.

“In general, iterative deepening search is the preferred uninformed search method when the state space is large and the depth of the solution is unknown.” (R&N)
Next Week

*Informed searches* … what if you know something about the solution?

What form should such knowledge take?

How should you use it?