Search

EECS 395/495
Intro to Artificial Intelligence
Doug Downey
(slides from Oren Etzioni, based on Stuart Russell, Dan Weld, Henry Kautz, and others)
What is Search?

Search is a class of techniques for systematically finding or constructing solutions to problems.

Example technique: generate-and-test.
Example problem: Combination lock.

1. Generate a possible solution.
2. Test the solution.
3. If solution found THEN done ELSE return to step 1.
Search thru a
Problem Space / State Space

Input:

- Set of states
- Operators [and costs]
- Start state
- Goal state [test]

Output:

- Path: start ⇒ a state satisfying goal test
- [May require shortest path]
Why is search interesting?

Many (all?) AI problems can be formulated as search problems!

Examples:
• Path planning
• Games
• Natural Language Processing
• Machine learning
• Genetic algorithms
Example: The 8-puzzle

- states?
- actions?
- goal test?
- path cost?
Example: The 8-puzzle

- **states?** locations of tiles
- **actions?** move blank left, right, up, down
- **goal test?** = goal state (given)
- **path cost?** 1 per move

[Note: optimal solution of \(n\)-Puzzle family is NP-hard]
Search Tree Example:
Fragment of 8-Puzzle Problem Space
Example: robotic assembly

**states?**: real-valued coordinates of robot joint angles
**parts of the object to be assembled**

**actions?**: continuous motions of robot joints

**goal test?**: complete assembly

**path cost?**: time to execute
Example: Romania

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

Formulate goal:
• be in Bucharest
  
Formulate problem:
• states: various cities
• actions: drive between cities
  
Find solution:
• sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest
Example: N Queens

Input:
- Set of states
- Operators [and costs]
- Start state
- Goal state (test)

Output
Implementation: states vs. nodes

A state is a (representation of) a physical configuration.

A node is a data structure constituting part of a search tree includes state, parent node, action, path cost \( g(x) \), depth.

The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.
Tree Search

Fringe = root node
Repeat while fringe non-empty
   Take n from Fringe
   If n is a goal
      break (we’re done)
   Else
      add n’s successors to Fringe
If n is a goal, return path to n
Otherwise return failure
A search strategy is defined by picking the order of node expansion.

Strategies are evaluated along the following dimensions:

- **Completeness**: does it always find a solution if one exists?
- **Time complexity**: number of nodes generated
- **Space complexity**: maximum number of nodes in memory
- **Optimality**: does it always find a least-cost solution?
- **Systematicity**: does it visit each state at most once?

Time and space complexity are measured in terms of:

- $b$: maximum branching factor of the search tree
- $d$: depth of the least-cost solution
- $m$: maximum depth of the state space (may be $\infty$)
Uninformed search strategies

Uninformed search strategies use only the information available in the problem definition.

- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search
Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!
Depth First Search

Maintain stack of nodes to visit

Evaluation

• Complete?
  Not for infinite spaces

• Time Complexity?
  $O(b^m)$

• Space Complexity?
  $O(m)$
  (though vanilla alg. in book has no backtracking, so $O(mb)$)
Breadth First Search

Maintain queue of nodes to visit

Evaluation

• **Complete?**
  Yes (assume \( b \) finite)

• **Time Complexity?**
  \( O(b^{d+1}) \)

• **Space Complexity?**
  \( O(b^{d+1}) \)
BFS: Memory Limitation

Suppose:

2 GHz CPU
1 GB main memory
100 cycles / expansion
5 bytes / node

200,000 expansions / sec
Memory filled in 100 sec ... < 2 minutes
Iterative deepening search

function \textsc{Iterative-Deepening-Search}(\textit{problem}) \textbf{return}s a solution, or failure

\begin{itemize}
  \item inputs: \textit{problem}, a problem
  \item for \textit{depth} $\leftarrow$ 0 to $\infty$ do
    \item result $\leftarrow$ \textsc{Depth-Limited-Search}(\textit{problem}, \textit{depth})
    \item if result $\neq$ cutoff then return result
\end{itemize}
Iterative deepening search / $l = 0$
Iterative deepening search / =1

Limit = 1
Iterative deepening search \( l = 2 \)

Limit = 2

Diagram of iterative deepening search with a limit of 2, showing the progression of the search through depth-limited searches up to this limit.
Iterative deepening search / $\leq 3$
Iterative deepening search

Number of nodes generated in a depth-limited search to depth $d$ with branching factor $b$:

$$N_{DLS} = b^0 + b^1 + b^2 + ... + b^{d-2} + b^{d-1} + b^d$$

Number of nodes generated in an iterative deepening search to depth $d$ with branching factor $b$:

$$N_{IDS} = (d+1)b^0 + d b^1 + (d-1)b^2 + ... + 3b^{d-2} + 2b^{d-1} + 1b^d$$

For $b = 10$, $d = 5$,

- $N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111$
- $N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$

Overhead = $(123,456 - 111,111)/111,111 = 11\%$
# Cost of Iterative Deepening

<table>
<thead>
<tr>
<th>$b$</th>
<th>ratio ID to DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>10</td>
<td>1.2</td>
</tr>
<tr>
<td>25</td>
<td>1.08</td>
</tr>
<tr>
<td>100</td>
<td>1.02</td>
</tr>
</tbody>
</table>
iterative deepening search

Complete? Yes
Time?
  • \((d+1)b^0 + d b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)\)
Space?
  • \(O(d)\)
Optimal?
  • Yes, if step cost = 1
Systematic?
  • No, but okay
Repeated states (Part II)

Failure to detect repeated states can turn a linear problem into an exponential one!
Repeated states (Part II)

More realistic case:

A -- B -- C
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
E     F     G
|     |     |     |
I     J     K
|     |     |     |
M     N     O
|     |     |     |
P     H     L
Repeated states (Part II)

Can save states but...then iterative deepening with DFS no longer takes $O(m)$ space!

What space is required?
Forwards vs. Backwards
vs. Bidirectional
Recap: Tree Search

Fringe = root node
Repeat while fringe non-empty
  Take \( n \) from Fringe
  If \( n \) is a goal
      break (we’re done)
  Else
      Expand \( n \): add \( n \)'s successors to Fringe

If \( n \) is a goal, return path to \( n \)
Otherwise return failure
Summary

Search problems

Search strategies

DFS, BFS, Iterative Deepening w/depth-limited search

Issue: All these methods are slow (blind)

Solution → add guidance (“heuristic estimate”)
           → “informed search”