Uninformed Search

Chapter 3

(Based on slides by Stuart Russell, Dan Weld, Oren Etzioni, Henry Kautz, and other UW-AI faculty)
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 $\Rightarrow$ 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
  • ...

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.
Search strategies

• 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?
  - Time Complexity?
  - Space Complexity?

Yes except for infinite spaces

\[ O(b^m) \]
\[ O(bm) \]

http://www.youtube.com/watch?v=dtoFAvtVE4U
Breadth First Search

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes (b is finite)
  - Time Complexity? $O(b^d)$
  - Space Complexity? $O(b^d)$

http://www.youtube.com/watch?v=z6IUnb9ktkE
Memory Limitation

• Suppose:
  2 GHz CPU
  1 GB main memory
  100 instructions / expansion
  5 bytes / node

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

function Iterative-Deepening-Search(problem) returns a solution, or failure

inputs: problem, a problem

for depth ← 0 to ∞ do
    result ← Depth-Limited-Search(problem, depth)
    if result ≠ cutoff then return result
Iterative deepening search $l = 0$
Iterative deepening search / $l = 1$

Limit = 1
Iterative deepening search \( l = 2 \)
Iterative deepening search $l = 3$
Iterative deepening search

- Number of nodes generated in a depth-limited search to depth $d$ with branching factor $b$:
  \[ N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d \]

- Number of nodes generated in an iterative deepening search to depth $d$ with branching factor $b$:
  \[ N_{IDS} = (d+1)b^0 + db^{1} + (d-1)b^{2} + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d \]

- 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 = \( (123,456 - 111,111)/111,111 = 11\% \)
iterative deepening search

• **Complete?** Yes

• **Time?**
  – \((d+1)b^0 + d b^1 + (d-1)b^2 + \ldots + b^d = O(b^{d+1})\)

• **Space?**
  – \(O(bd)\)

• **Optimal?**
  – Yes, if step cost = 1

• **Systematic?**
### Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{C*/\epsilon})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{C*/\epsilon})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Forwards vs. Backwards
vs. Bidirectional
Problem

• All these methods are slow (blind)

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