Solving Problems by Searching

CS 486/686: Introduction to Artificial Intelligence
Outline

• Problem solving agents and search
• Examples
• Properties of search algorithms
• Uniformed search
  - Breadth first
  - Depth first
  - Iterative deepening
Introduction

- Search was one of the first topics studied in AI
  - Newell and Simon (1961) *General Problem Solver*

- Central component to many AI systems
  - Automated reasoning, theorem proving, robot navigation, scheduling, game playing,...
Defining a Search Problem

- **State space** $S$: all possible configs. of the domain
- **Initial state** $s_0 \in S$: the start state
- **Goal states** $G \subseteq S$: the set of end states
  - **Goal test**: check if we are in a goal state
- **Operators** $A$: actions available
  - Often defined in terms of mappings from state to successor state
Defining a Search Problem

- **Path**: a sequence of states and operators
- **Path cost** $c$: a number associated with any path
- **Solution**: a path from $s_0$ to $s_G \in G$
- **Optimal solution**: a path with minimum cost
Example: Traveling in Romania
Examples of Search Problems

- States:
- Initial State:
- Operators:
- Goal test:
- Path cost:

- States:
- Initial State:
- Operators:
- Goal test:
- Path cost:
Examples of Search Problems
Our Definition Excludes...

- Chance
- Continuous states
- Partial Observability
- Adversaries
- All of the above
Representing Search

• Search graph
  - Vertices correspond to states
  - Edges correspond to operators

• We search for a solution by building a search tree and traversing it to find a goal state
Data Structures: Search Node

- State
- Parent node and operator applied to parent to reach current node
- Cost of path so far
- Depth of node
Expanding Nodes

• Expanding a node
  - Applying all legal operators to the state contained in the node
  - Generating nodes for all corresponding successor states
Expanding Nodes

(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu
Generic Search Algorithm

- Initialize with initial state of the problem
- Repeat
  - If no candidate nodes can be expanded return failure
  - Choose leaf node for expansion, according to search strategy
  - If node contains goal state, return solution
  - Otherwise, expand the node. Add resulting nodes to the tree
Implementation Details

• Need to keep track of nodes to be expanded (fringe)

• Implement using a queue:
  - Insert node for initial state
  - Repeat
    - If queue is empty, return failure
    - Dequeue a node
      - If node contains goal state, return solution
      - Expand node

• Search algorithms differ in their queuing function!
Breadth First Search

- All nodes on a given depth are expanded before any nodes on next level are expanded.
- Implemented with a **FIFO** queue
Key Properties

• **Completeness**: Is the alg. guaranteed to find a solution if the solution exists?

• **Optimality**: Does the alg. find the optimal solution?

• **Time complexity**: How many operations are needed?

• **Space complexity**: How much storage is needed?

• Other desirable properties
  - Can the alg. return an intermediate solution?
  - Can an adequate solution be refined or improved?
Search Performance

- Evaluated in terms of 2 characteristics
  - **Branching factor of state space**: how many operators can be applied at any time?
  - **Solution depth**: how long is the path to the closest solution?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Branching factor</td>
</tr>
<tr>
<td>d</td>
<td>Depth of shallowest goal node</td>
</tr>
<tr>
<td>m</td>
<td>Maximum length of any path in the state space</td>
</tr>
</tbody>
</table>
Judging BFS

- **Good news**
  - Complete (if b is finite)
  - Optimal (if all costs are the same)

- **Bad news**
  - Exponential time complexity: $O(b^{d+1})$
    - A problem with all uninformed search methods
  - Exponential space complexity: $O(b^{d+1})$
    - Horrible!
Making BFS Always Optimal

- Uniform cost search algorithm
  - Use a priority queue instead of a simple queue
  - Insert nodes in increasing order of the cost of path so far
  - Guaranteed to find optimal solution
Uniform Cost Search

\[ C^* \text{ is cost of optimal solution} \]

\[ \varepsilon \text{ is minimum action cost} \]

**Time:**
\[ O\left(b^{1+\text{floor}(C^*/\varepsilon)}\right) \]

**Space:**
\[ O\left(b^{1+\text{floor}(C^*/\varepsilon)}\right) \]
Depth First Search

- Deepest node in current fringe of the search tree is expanded first
- Implemented with a stack
Judging Depth First Search

- **Bad news**
  - Not complete: might get stuck going down a long path
  - Not optimal: might return a solution at greater depth than another solution
  - Time complexity: $O(b^m)$
    - $m$ might be much larger than $d$

- **Good news**
  - Space complexity: $O(bm)$
Depth Limited Search

- Search depth-first, but terminate path if
  - a goal is found, or
  - maximum depth, l, is reached.

- How do you set l?
- What happens if l=1?
Depth Limited Search

- **Good news**
  - Always terminates
  - Space: $O(bl)$

- **Bad news**
  - Not complete
    - Goal depth might be deeper than $l$
  - Time: $O(b^l)$
Iterative Deepening

- Depth limited search, but increase the limit each iteration

Figure 3.16  Four iterations of iterative deepening search on a binary tree.
Judging Iterative Deepeing

• Bad news
  - Time: $O(b^d)$

• Good news
  - Complete (like BFS)
  - Optimal
  - Space: $O(bd)$
Iterative Deepening

• Isn’t IDS very wasteful?
  - Expanding the same nodes multiple times

• Insight
  - Most nodes are found in bottom level of the tree
Revisiting States

- What if we revisit a state that has already been expanded?

- What if we visit a state that is already in the queue?
Revisiting States

- Maintain a closed list to store every expanded node
  - More efficient on problems with many repeated states
  - Worst-case time and space are $O(S)$
    - $S$ is number of states

- Allowing states to be re-expanded can produce a better solution
  - What should you do?
Summary

• Assumes no knowledge about the problem
  - This makes them general but expensive

• Variety of uninformed search strategies
  - Mainly differ in the order in which they consider states

<table>
<thead>
<tr>
<th>Criteria</th>
<th>BFS</th>
<th>Uniform</th>
<th>DFS</th>
<th>DLS</th>
<th>IDS</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>$O(b + 1 + \text{floor}(C*/\epsilon))$</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>$O(b + 1 + \text{floor}(C*/\epsilon))$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(b(d)$</td>
</tr>
<tr>
<td>Optimal</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Questions?

- Next class: Informed search