Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods (if time)
  - Greedy Search

Review: Rational Agents

- An agent is an entity that perceives and acts.
- A rational agent selects actions that maximize its utility function.
- Characteristics of the percepts, environment, and action space dictate techniques for selecting rational actions.

Search -- the environment is:
- fully observable, single agent, deterministic, episodic, discrete

Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - Do not consider future consequences of their actions
  - Act on how the world IS
- Can a reflex agent be rational?

Famous Reflex Agents

- [Image of a robot]
Goal Based Agents

- Goal-based agents:
  - Plan ahead
  - Ask "what if"
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Act on how the world WOULD BE

Search Problems

- A search problem consists of:
  - A state space
  - A successor function
  - A start state and a goal test

Example: Romania

- State space:
  - Cities
- Successor function:
  - Go to adjacent city with cost = dist
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

State Space Graphs

- State space graph:
  - Each node is a state
  - The successor function is represented by arcs
  - Edges may be labeled with costs
  - We can rarely build this graph in memory (so we don’t)

State Space Sizes?

- Search Problem: Eat all of the food
  - Pacman positions: 10 x 12 = 120
  - Pacman facing: up, down, left, right
  - Food Count: 30
  - Ghost positions: 12

Search Trees

- A search tree:
  - Start state at the root node
  - Children correspond to successors
  - Nodes contain states, correspond to PLANS to those states
  - Edges are labeled with actions and costs
  - For most problems, we can never actually build the whole tree
Searching with a Search Tree

- Search:
  - Expand out possible plans
  - Maintain a fringe of unexpanded plans
  - Try to expand as few tree nodes as possible

General Tree Search

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

- Main question: which fringe nodes to explore?

Example: Tree Search

State Graphs vs. Search Trees

We construct both on demand – and we construct as little as possible.

Review: Depth First Search

Strategy: expand deepest node first
Implementation: Fringe is a LIFO queue (a stack)
Review: Breadth First Search

**Strategy:** expand shallowest node first

**Implementation:** Fringe is a FIFO queue

---

**Search Algorithm Properties**

- Complete? Guaranteed to find a solution if one exists?
- Optimal? Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

**Variables:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of states in the problem</td>
</tr>
<tr>
<td>$b$</td>
<td>The maximum branching factor $B$ (the maximum number of successors for a state)</td>
</tr>
<tr>
<td>$C^*$</td>
<td>Cost of least cost solution</td>
</tr>
<tr>
<td>$d$</td>
<td>Depth of the shallowest solution</td>
</tr>
<tr>
<td>$m$</td>
<td>Max depth of the search tree</td>
</tr>
</tbody>
</table>

---

**Comparisons**

- When will BFS outperform DFS?
- When will DFS outperform BFS?

---

**Iterative Deepening**

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less.
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.
   ... and so on.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>Y</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y*</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>ID</td>
<td>Y</td>
<td>Y*</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
</tbody>
</table>

---

**Costs on Actions**

Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

---
Uniform Cost Search

Expand cheapest node first:
Fringe is a priority queue

Expansion order:
(S, p, d, b, e, a, f, e, G)

Algorithm | Complete | Optimal | Time | Space
---|---|---|---|---
DFS (w/ Path Checking) | Y | N | O(b^m) | O(bm)
BFS | Y | Y* | O(b^d) | O(b^d)
UCS | Y* | Y | O(b^{d*/ε}) | O(b^{d*/ε})

C*ε tiers