CIS 471/571: Artificial Intelligence
Fall 2012

Lecture 2: Uninformed Search
9/26/2012

Daniel Lowd
Most slides are by Luke Zettlemoyer, John DeNero, Dan Klein, Stuart Russell or Andrew Moore

Announcements

- **Project 0: Python Tutorial**
  - Posted online now
  - Due Thursday, September 27th

- **Project 1A: Uninformed Search**
  - Posted online now
  - Due next week: Thursday, October 3rd
  - Start early and ask questions.

- Please post questions (and answers!) on Piazza.
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
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

- A solution is a sequence of actions (a plan) which transforms the start state to a goal state
Example: Romania

- State space:
  - Cities
- Successor function:
  - Go to adj 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:
  \(10 \times 12 = 120\)
- 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

```
def TREE-SEARCH(problem, strategy)
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- **Important ideas:**
  - Fringe
  - Expansion
  - Exploration strategy

- **Main question:** which fringe nodes to explore?
Example: Tree Search

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

State Graphs vs. Search Trees

Each NODE in in the search tree is an entire PATH in the problem graph.

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)

Expansion ordering:
(d,b,a,c,a,e,h,p,q,q,r,f,c,a,G)
Review: Breadth First Search

**Strategy:** expand shallowest node first

**Implementation:**
Fringe is a FIFO queue

Expansion order:
(S,d,e,p,b,c,e,h,r,q,a,a,h,r,p,q,f,p,q,f,q,c,G)
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></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 w/ Path Checking</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^l)</td>
<td>O(b^l)</td>
</tr>
<tr>
<td>ID</td>
<td>Y</td>
<td>Y*</td>
<td>O(b^d)</td>
<td>O(bd)</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:

$(S, p, d, b, e, a, r, f, e, G)$
# Uniform Cost Search

<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 (\text{w/ Path Checking})</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^d))</td>
<td>(O(b^d))</td>
</tr>
<tr>
<td>UCS</td>
<td>Y*</td>
<td>Y</td>
<td>(O(b^{C^*/\varepsilon}))</td>
<td>(O(b^{C^*/\varepsilon}))</td>
</tr>
</tbody>
</table>

\(C^*/\varepsilon\) tiers