Announcements

- Project 0: Python Tutorial
  - Due tomorrow!
  - There is a lab Wednesday from 3pm-5pm in Soda 275
  - The lab time is optional, but P0 itself is not
  - On submit, you should get email from the autograder

- Project 1: Search
  - On the web today
  - Start early and ask questions. It’s longer than most!

- Self-Diagnostic on web
- Sections: can go to any, but have priority in your own

CS 188: Artificial Intelligence
Fall 2011

Lecture 2: Queue-Based Search
8/30/2011

Dan Klein – UC Berkeley
Multiple slides from Stuart Russell, Andrew Moore

Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods (part review for some)
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods (new for all)
  - Greedy Search

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
  - Consider how the world WOULD BE

Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world’s current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS
  - Can a reflex agent be rational?

Search Problems

- A search problem consists of:
  - A state space
  - A successor function (with actions, costs)
  - 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

- "N", 1.0
- "E", 1.0
- "W", 1.0
- "S", 1.0

[demo: reflex optimal / loop ]

[demo: plan fast / slow ]
Example: Romania

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

State Space Graphs

- State space graph: A mathematical representation of a search problem
  - For every search problem, there's a corresponding state space graph
  - The successor function is represented by arcs
- We can rarely build this graph in memory (so we don't)

What's in a State Space?

A state space is a mathematical representation of a search problem. It consists of:

- States: The possible configurations of the problem
- Actions: The operations that can change the state
- Successor function: How actions change states
- Goal test: How to determine if a state is a goal

What's in a State Space?

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?
    - 120x(2^30)x(12)^4
  - States for pathing?
    - 120
  - States for eat-all-dots?
    - 120x(2^30)

Search Trees

- A search tree:
  - This is a "what if" tree of plans and outcomes
  - Start state at the root node
  - Children correspond to successors
  - Nodes contain states, correspond to PLANS to those states
  - For most problems, we can never actually build the whole tree

Another 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

Review: Depth First Search

Review: Breadth First Search

Search Algorithm Properties

<table>
<thead>
<tr>
<th>Complete?</th>
<th>Guaranteed to find a solution if one exists?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal?</td>
<td>Guaranteed to find the least cost path?</td>
</tr>
<tr>
<td>Time complexity?</td>
<td></td>
</tr>
<tr>
<td>Space complexity?</td>
<td></td>
</tr>
</tbody>
</table>

Variables:

- \( n \) Number of states in the problem (huge)
- \( b \) The average branching factor \( B \) (the average number of successors)
- \( C^* \) Cost of least cost solution
- \( s \) Depth of the shallowest solution
- \( m \) Max depth of the search tree
DFS

- Infinite paths make DFS incomplete...
- How can we fix this?

<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>N</td>
<td>N</td>
<td>Infinite</td>
<td>Infinite</td>
</tr>
</tbody>
</table>

With cycle checking, DFS is complete.*

When is DFS optimal?

Comparisons

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

Iterative Deepening

Iterative deepening: BFS using 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>$O(b^{m+1})$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N</td>
<td>$O(b^{m+1})$</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. We will quickly cover an algorithm which does find the least-cost path.

BFS

- When is BFS optimal?

Algorithm Complete Optimal Time Space
<table>
<thead>
<tr>
<th>DFS w/ Path Checking</th>
<th>Y</th>
<th>N</th>
<th>$O(b^{m+1})$</th>
<th>$O(bm)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N*</td>
<td>$O(b^{m+1})$</td>
<td>$O(bm)$</td>
</tr>
</tbody>
</table>

* Or graph search – next lecture.
Uniform Cost Search

Expand cheapest node first:
Fringe is a priority queue (priority: cumulative cost)

Cost contours

Priority Queue Refresher

- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:
  - `pq.push(key, value)` inserts (key, value) into the queue.
  - `pq.pop()` returns the key with the lowest value, and removes it from the queue.
- You can decrease a key’s priority by pushing it again
- Unlike a regular queue, insertions aren’t constant time, usually $O(\log n)$
- We’ll need priority queues for cost-sensitive search methods

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</td>
<td>N</td>
<td>$O(b^{m+1})$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N</td>
<td>$O(b^{m+1})$</td>
<td>$O(b)$</td>
</tr>
<tr>
<td>UCS</td>
<td>Y*</td>
<td>Y</td>
<td>$O(b^{m+1})$</td>
<td>$O(b^{m+1})$</td>
</tr>
</tbody>
</table>

* UCS can fail if actions can get arbitrarily cheap

Uniform Cost Issues

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every direction
  - No information about goal location

Search Heuristics

- Any estimate of how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance

Heuristics
Best First / Greedy Search

- Expand the node that seems closest...

- What can go wrong?

Best First / Greedy Search

- A common case:
  - Best-first takes you straight to the (wrong) goal

- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking

- Like DFS in completeness (finite states w/ cycle checking)

Search Gone Wrong?