Announcements

- Project 0: Python Tutorial
  - Due tomorrow!
  - There is a lab tomorrow from 1pm-3pm 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!

- Other
  - Section 107 was opened up, Fridays 1-2pm
  - My OHs Monday were in the lab, Thursday back in 711 Soda
  - GSI OHs on web site

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

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
  - Act on how the world IS
  - Can a reflex agent be rational?

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 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!)

State Space Sizes?

- Search Problem: Eat all of the food
  - Pacman positions: 10 x 12 = 120
  - Food count: 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?

Detailed pseudocode is in the book!
Nodes in state space graphs are problem states
- Represent an abstracted state of the world
- Have successors, can be goal / non-goal, have multiple predecessors
Nodes in search trees are plans
- Represent a plan (sequence of actions) which results in the node’s state
- Have a problem state and one parent, a path length, a depth & a cost
- The same problem state may be achieved by multiple search tree nodes

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

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

Algorithm Complete Optimal Time Space
DFS Depth First Search N N Infinite Infinite

BFS

- When is BFS optimal?

Algorithm Complete Optimal Time Space
DFS w/ Path Checking Y N O(b^{s+1}) O(bs)
BFS Y N* O(b^{s+1}) O(bs)

Iterative Deepening

Iterative deepening uses DFS as a subroutine:
1. Do a DFS which only searches 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.

Algorithm Complete Optimal Time Space
DFS w/ Path Checking Y N O(b^{s+1}) O(bs)
BFS Y N* O(b^{s+1}) O(bs)

Comparisons

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

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.
Uniform Cost Search

Expand cheapest node first:
Fringe is a priority queue

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 w/ Push Checking</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>$O(b^{m+2})$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>UCS</td>
<td>Y*</td>
<td>Y</td>
<td>$O(b^{s+1})$</td>
<td>$O(b^s)$</td>
</tr>
<tr>
<td>C* tiers</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^{C*/\varepsilon})$</td>
<td>$O(b^{C*/\varepsilon})$</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

Single Num Distance to Ubud

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yogyakarta</td>
<td>160</td>
</tr>
<tr>
<td>Denpasar</td>
<td>150</td>
</tr>
<tr>
<td>Ubud</td>
<td>140</td>
</tr>
<tr>
<td>Bali</td>
<td>130</td>
</tr>
<tr>
<td>Denpasar</td>
<td>120</td>
</tr>
<tr>
<td>Ubud</td>
<td>110</td>
</tr>
<tr>
<td>Yogyakarta</td>
<td>100</td>
</tr>
<tr>
<td>Denpasar</td>
<td>90</td>
</tr>
<tr>
<td>Ubud</td>
<td>80</td>
</tr>
<tr>
<td>Denpasar</td>
<td>70</td>
</tr>
<tr>
<td>Ubud</td>
<td>60</td>
</tr>
</tbody>
</table>

[demo: search demo empty]
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?