Introduction to Artificial Intelligence

V22.0472-001 Fall 2009

Lecture 2: Queue-based search

Rob Fergus – Dept of Computer Science, Courant Institute, NYU

Many slides from
Dan Klein, Stuart Russell and Andrew Moore

Announcements

- Python lab tomorrow
  - 7-8pm in this room
  - Please install Python before you come
  - Bring textbook
  - Windows/Mac/Linux??

- First assignment will be posted on Wed
  - Small written component
  - Bigger coding component

Today

- Agents that Plan Ahead
- Search Problems
- Uniformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods
  - Greedy Search
  - A* Search [WEDNESDAY]

Reflex Agents

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

Goal Based Agents

- Goal-based agents:
  - Plan ahead
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions

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
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 labeled with states, correspond to PLANS to those states
  - For most problems, can never actually build the whole tree
    - So, have to find ways of using only the important parts of the tree!

State Space Graphs

- There's some big graph in which
  - Each state is a node
  - Each successor is an outgoing arc
- Important: For most problems we could never actually build this graph
- How many states in Pacman?

Example: Romania

Another Search Tree

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

States vs. Nodes

- Problem graphs have problem states
  - Represent an abstracted state of the world
  - Have successors, can be goal / non-goal, have multiple predecessors

- Search trees have search nodes
  - Represent a plan (path) which results in the node's state
  - Have single parents, a path length and cost, point to a problem state
  - Expand uses successor function to create new search tree nodes
  - The same problem state may be in multiple search tree nodes

General Tree Search

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?
**Example: Tree Search**

**State Graphs vs Search Trees**

Each node in the search tree is an entire path in the problem graph.

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

**Depth First Search**

Strategy: expand deepest node first
Implementation: Fringe is a LIFO stack

**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><em>n</em></td>
<td>Number of states in the problem</td>
</tr>
<tr>
<td><em>b</em></td>
<td>The average branching factor B (the average number of successors)</td>
</tr>
<tr>
<td><em>B</em></td>
<td>Cost of least cost solution</td>
</tr>
<tr>
<td><em>s</em></td>
<td>Depth of the shallowest solution</td>
</tr>
<tr>
<td><em>m</em></td>
<td>Max depth of the search tree</td>
</tr>
</tbody>
</table>

**DFS**

<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>

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

- With cycle checking, DFS is complete.

![DFS Diagram]

<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(hm))</td>
</tr>
</tbody>
</table>

- When is DFS optimal?

BFS

- When is BFS optimal?

![BFS Diagram]

<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(hm))</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N*</td>
<td>(O(b^{s+1}))</td>
<td>(O(hs))</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. (DFS gives up on any path of length 2)
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.

![Iterative Deepening Diagram]

<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(hm))</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N*</td>
<td>(O(b^{s+1}))</td>
<td>(O(hs))</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.

![Costs on Actions Diagram]

Uniform Cost Search

Expand cheapest node first:
Fringe is a priority queue

![Uniform Cost Search Diagram]
**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 promote or demote keys by resetting their priorities.
- Unlike a regular queue, insertions into a priority queue are not constant time, usually $O(\log n)$.
- We’ll need priority queues for most cost-sensitive search methods.

**Uniform Cost Search**

- What will UCS do for this graph?

```
1 0 0
0 0 0
1 0 1
```

- What does this mean for completeness?

**Uniform Cost Problems**

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

**Heuristics**

- Expand the node that seems closest…

**Best First / Greedy Search**

- 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?

Extra Work?

• Failure to detect repeated states can cause exponentially more work. Why?

Graph Search

• In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)

• Very simple fix: never expand a state type twice

• Can this wreck completeness? Why or why not?
• How about optimality? Why or why not?
Some Hints

- Graph search is almost always better than tree search (when not?)

- Fringes are sometimes called “closed lists” – but don’t implement them with lists (use sets)!

- Nodes are conceptually paths, but better to represent with a state, cost, and reference to parent node

Best First Greedy Search

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Best-First Search</td>
<td>Y*</td>
<td>N</td>
<td>O(b^m)</td>
<td>O(b^m)</td>
</tr>
</tbody>
</table>

- What do we need to do to make it complete?
- Can we make it optimal? Next class!