CSE 473: Artificial Intelligence
Spring 2014

Hanna Hajishirzi
Search with Cost & Heuristics

slides from
Dan Klein, Stuart Russell, Andrew Moore, Dan Weld, Pieter Abbeel, Luke Zettlemoyer
Announcement

- PS1 will be on the web soon!
  - Start early
Recap: Search

- **Search problem:**
  - States (configurations of the world)
  - Successor function; drawn as a graph
  - Start state and goal test

- **Search tree:**
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
General Tree Search

- **Search Algorithms:**
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)

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

- **Main question:** which fringe nodes to explore?
Outline

- Uninformed Search Methods (part review for some)
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search

- Heuristic Search Methods (new for all)
  - Best First / Greedy Search
  - A*
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
Review: Breadth First Search

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>n</th>
<th>Number of states in the problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>The maximum branching factor B</td>
</tr>
<tr>
<td></td>
<td>(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>
Infinite paths make DFS incomplete…
- How can we fix this?
- Check new nodes against path from S

Infinite search spaces still a problem
- If the left subtree has unbounded depth

<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>No</td>
<td>No</td>
<td>Infinite</td>
<td>Infinite</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>Depth First Search</td>
<td>No</td>
<td>No</td>
<td>Infinite</td>
</tr>
</tbody>
</table>
DFS

Algorithm
DFS w/ Path Checking

Complete
Y if finite

Optimal
N

Time
O(b^m)

Space
O(bm)

Diagram:
- DFS with Path Checking
- m tiers: 1 node, b nodes, b^2 nodes, b^m nodes
### BFS

<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>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
</tbody>
</table>

**Diagram:**
- The diagram illustrates a tree structure with $d$ tiers.
- There is 1 node at the top, $b$ nodes at the second tier, $b^2$ nodes at the third tier, $b^d$ nodes at the $d$th tier, and $b^m$ nodes at the bottom.
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>w/ Path Checking</td>
<td>Y</td>
<td>N</td>
<td>O(b^m)</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>ID</td>
<td>Y</td>
<td>Y*</td>
<td>O(b^d)</td>
<td>O(bd)</td>
</tr>
</tbody>
</table>
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
Uniform Cost Search

- Generalization of breadth-first search
- *Priority* queue of nodes to be explored
- Cost function $f(n)$ applied to each node

Add initial state to priority queue
While queue not empty
  Node = head(queue)
  If goal?(node) then return node
  Add children of node to queue
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.

 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

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 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>$O(b^d)$</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
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
Uniform Cost: Pac-Man

- Cost of 1 for each action
- Explores all of the states, but one
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

Best first with \( f(n) = \) heuristic estimate of distance to goal
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)
To Do:

- Look at the course website:
- Do the readings (Ch 3)
- Do PS0 if new to Python
- Start PS1, when it is posted
  - START PS1 ASAP