Lecture 3: Search
1/24/2006

Dan Klein – UC Berkeley
Many slides from either Stuart Russell or Andrew Moore

Today

- Formulating Search Problems
- Uniformed Search
  - Depth First Search
  - Breadth First Search
  - Uniform Cost Search
- Properties of Search Algorithms

Example: Romania

Single State Problems

- A search problem is defined by four items:
  - Initial state: e.g. Arad
  - Successor function S(x) = set of action–state pairs:
    e.g., S(Arad) = {<Arad → Zerind, Zerind>, ... }
  - Goal test, can be
    - explicit, e.g., x = Bucharest
    - implicit, e.g., Checkmate(x)
  - Path cost (additive)
    - e.g., sum of distances, number of actions executed, etc.
    - c(a,b) is the step cost, assumed to be ≥ 0
- A solution is a sequence of actions leading from the initial state to a goal state

Search Gone Wrong?
Example: Vacuum World

- Can represent problem as a state graph
  - Nodes are states
  - Arcs are actions
  - Arc weights are costs

Example: 8-Puzzle

- What are the states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

Example: N-Queens

- What are the states?
- What is the start?
- What is the goal?
- What are the actions?
- What should the costs be?

Example: Assembly

- What are the states?
- What is the goal?
- What are the actions?
- What should the costs be?

Example: Romania

A Search Tree

- Search trees:
  - Represent the branching paths through a state graph.
  - Usually much larger than the state graph.
  - Can a finite state graph give an infinite search tree?
Tree Search

- Basic solution method for graph problems
  - Offline simulated exploration of state space
  - Searching a model of the space, not the real world

### Function: Tree-Search

```plaintext
Function: Tree-Search(problem, strategy) returns a solution, or failure
    fringe ← Insert(Make-Node(initial-state, problem), fringe)
    loop do
        if fringe is empty then return failure
        node ← Remove-Front(fringe)
        if Goal-Test(node, problem) then return Solution(node)
        fringe ← Insert-All(Expand(node, problem), fringe)
        fringe ← Insert(node, fringe)
    end
end
```

Trees vs. Nodes

- Problem graphs have problem states
  - Have successors

- Search trees have search nodes
  - Have parents, children, depth, path cost, etc.
  - Expand uses successor function to create new search tree nodes
  - The same problem state may be in multiple search tree nodes

A Search Graph

How do we get from S to G? And what’s the smallest possible number of transitions?

Depth First Search

Expand deepest node first:
- Fringe is a LIFO stack

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

Infinite paths make DFS incomplete…
How can we fix this?

Algorithm | Complete | Optimal | Time | Space
--- | --- | --- | --- | ---
DFS | N | N | Infinite | Infinite

DFS

With cycle checking, DFS is complete.

Algorithm | Complete | Optimal | Time | Space
--- | --- | --- | --- | ---
DFS | Y | N | O(b^{m+1}) | O(bm)

Breadth First Search

Expand shallowest node first:
Fringe is a FIFO queue

Search Tiers

Algorithm | Complete | Optimal | Time | Space
--- | --- | --- | --- | ---
DFS | Y | N | O(b^{m+1}) | O(bm)
BFS | Y | N* | O(b^{m+1}) | O(df)

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

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.setPriority(key, value)` inserts (key, value) into the queue.
  - `pq.dequeue()` 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?

What does this mean for completeness?

Iterative Deepening

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

Algorithm Complete Optimal Time Space
DFS w/ Path Checking Y N O(bm+1) O(kw)
BFS Y N O(bm) O(kw)
UCS Y* Y O(C* bC*/(tiers)) O(kw)

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?)

**Uniform Cost Issues**

- Where will uniform cost explore?
- Why?
- What is wrong here?

**Greedy Search**

- Expand the node that seems closest…
- What can go wrong?

**Greedy Search**

- Very simple fix: never expand a node twice

**Euclidian Heuristic**
### 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>

- Can we make it optimal? Next class!