Search

Luke Zettlemoyer

Slides from Dan Klein, Stuart Russell, Andrew Moore
Outline

- 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)
    - Best First / Greedy Search
Review: Rational Agents

- An agent is an entity that perceives and acts.
- A rational agent selects actions that maximize its utility function.
- Characteristics of the percepts, environment, and action space dictate techniques for selecting rational actions.

Search -- the environment is:

- fully observable, single agent, deterministic, episodic, discrete
Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - Do not consider the future consequences of their actions
  - Act on how the world IS
- Can a reflex agent be rational?
- Can a non-rational agent achieve goals?
Famous Reflex Agents
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

“N”, 1.0

“E”, 1.0
Example: Romania

- **State space:**
  - Cities

- **Successor function:**
  - Go to adj city with cost = dist

- **Start state:**
  - Arad

- **Goal test:**
  - Is state == Bucharest?

- **Solution?**
State Space Graphs

- State space graph:
  - Each node is a state
  - The successor function is represented by arcs
  - Edges may be labeled with costs
- 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
- Pacman facing: up, down, left, right
- Food Count: 30
- Ghost positions: 12
A search tree:

- Start state at the root node
- Children correspond to successors
- Nodes contain states, correspond to PLANS to those states
- Edges are labeled with actions and costs
- For most problems, we can never actually build the whole tree
Example: Tree Search

State Graph:

What is the search tree?
State Graphs vs. Search Trees

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

We construct both on demand – and we construct as little as possible.
Building Search Trees

- **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?

```plaintext
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

Detailed pseudocode is in the book!
**Strategy**: expand deepest node first

**Implementation**: Fringe is a LIFO queue (a stack)
Review: Depth First Search

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>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 maximum branching factor $B$ (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>
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

$m$ tiers

1 node
b nodes
$b^2$ nodes
$b^m$ nodes

Algorithm Complete Optimal Time Space
--- --- --- --- ---
DFS w/ Path Checking Y N $O(b^m)$ $O(bm)$

* Or graph search – next lecture.
**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</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>

- **DFS** with Path Checking
- **BFS**

![Diagram showing BFS algorithm with different tiers and nodes](image)

- 1 node
- $b$ nodes
- $b^2$ nodes
- $b^d$ nodes
- $b^m$ nodes
Comparisons

- When will BFS outperform DFS?
- When will DFS outperform BFS?
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^n)</td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td>O(b^d)</td>
<td>O(b^d)</td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td></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

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($b^{C*/\theta}$)</td>
<td>O($b^{C*/\theta}$)</td>
</tr>
</tbody>
</table>

**Diagram:**

- **$C^{*/\theta}$ tiers**
- **Nodes and Edge Representation**

- **Algorithm:**
  - **DFS with Path Checking**
  - **BFS**
  - **UCS**
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

Expand closest node first: Fringe is a priority queue
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
- Get started on PS1, when it is posted
Search Gone Wrong?

Start: Haugesund, Rogaland, Norway
End: Trondheim, Sør-Trøndelag, Norway
Total Distance: 2713.2 Kilometers
Estimated Total Time: 47 hours, 31 minutes