CSE 473: Artificial Intelligence
Autumn 2014

Problem Spaces & Search

Dan Weld

With slides from
Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

Logistics

- Piazza
- PS0 Due for optional grading end of Wed 10/1

Outline

- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods
  - Best First / Greedy Search

Agent vs. Environment

- 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.

Actions? Percepts?

Recommender System
Types of Agents

- Reflex
- Goal oriented
- Utility-based

Famous Reflex 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 thru a Problem Space (aka State Space)

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]
- Output:
  - Path: start \(\rightarrow\) a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need a state that passes test]

Example: Simplified Pac-Man

- Input:
  - A state space
  - A successor function
  - A start state
  - A goal test
- Output:

Ex: Route Planning: Arad \(\rightarrow\) Bucharest

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:
**Ex: Blocks World**

- **Input:**
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- **Output:**
  - Partially specified plans
  - Plan modification operators
  - The null plan (no actions)
  - A plan which provably achieves
  - The desired world configuration

**Multiple Problem Spaces**

**Real World**
- States of the world (e.g. block configurations)
- Actions (take one world-state to another)

**Robot’s Head**

- **Problem Space 1**
  - PS states: models of world states
  - Operators: models of actions
- **Problem Space 2**
  - PS states: partially spec. plan
  - Operators: plan modification ops

**Algebraic Simplification**

- **Input:**
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- **Output:**

**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 configurations: \(2^{30}\)
- Ghost1 positions: 12
- Ghost 2 positions: 11

\(120 \times 12 \times 2^{30} = 6.8 \times 10^{13}\)

**Search Methods**

- **Blind Search**
  - Depth first search
  - Breadth first search
  - Iterative deepening search
  - Uniform cost search
- **Local Search**
- **Informed Search**
- **Constraint Satisfaction**
- **Adversary Search**
Search Trees

- 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 denotes an entire PATH in the problem graph.
- We construct both on demand – and we construct as little as possible.

States vs. Nodes

- 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

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
  - which fringe node to expand next?

Detailed pseudocode is in the book!
Review? Depth First Search

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

Review? 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>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>

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>No</td>
<td>No</td>
<td>Infinite</td>
<td>Infinite</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
**DFS**

- Complete: \( Y \) if finite
- Optimal: \( N \)
- Time: \( O(b^m) \)
- Space: \( O(bm) \)

**BFS**

- Complete: \( Y \)
- Optimal: \( Y \)
- Time: \( O(b^d) \)
- Space: \( O(b^d) \)

---

**Memory a Limitation?**

- Suppose:
  - 4 GHz CPU
  - 32 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node
  - 40 M expansions / sec
  - Memory filled in 160 sec ... 3 min

**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.

**Cost of Iterative Deepening**

<table>
<thead>
<tr>
<th>( b )</th>
<th>ratio ID to DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>10</td>
<td>1.2</td>
</tr>
<tr>
<td>25</td>
<td>1.08</td>
</tr>
<tr>
<td>100</td>
<td>1.02</td>
</tr>
</tbody>
</table>

* Or graph search – next lecture.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Puzzle</td>
<td>BFS</td>
<td>Iter. Deep.</td>
<td>Nodes</td>
</tr>
<tr>
<td>2x2x2 Rubik’s</td>
<td>$10^5$</td>
<td>.01 sec</td>
<td>$10^5$</td>
</tr>
<tr>
<td>2x2x2 Rubik’s</td>
<td>$10^6$</td>
<td>.2 sec</td>
<td>$10^6$</td>
</tr>
<tr>
<td>15 Puzzle</td>
<td>$10^{13}$</td>
<td>6 days</td>
<td>$10^{17}$</td>
</tr>
<tr>
<td>3x3x3 Rubik’s</td>
<td>$10^{19}$</td>
<td>68k yrs</td>
<td>$10^{20}$</td>
</tr>
<tr>
<td>24 Puzzle</td>
<td>$10^{25}$</td>
<td>12B yrs</td>
<td>$10^{37}$</td>
</tr>
</tbody>
</table>

**Why the difference?**
- Rubik has higher branching factor
- 15 puzzle has greater depth

Slide adapted from Richard Korf presentation.