CSE 473: Artificial Intelligence
Autumn2016

Problem Spaces & Search

Dan Weld

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

Logistics

- Read Ch 3
- Do PS0 by Monday (should be easy)
- Start PS1 (harder!)
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.
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

Types of Environments

- Fully observable vs. partially observable
- Single agent vs. multiagent
- Deterministic vs. stochastic
- Episodic vs. sequential
- Discrete vs. continuous
Search thru a Problem Space (aka State Space)

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

- **Output:**
  - Path: start ⇒ 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
    - ![State Space Diagram]
  - Successor function
    - ![Successor Function Diagram]
  - A start state
  - A goal test

- **Output:**
Ex: Route Planning: Arad → Bucharest

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

**Output:**

Different operators may be applicable in different states.

---

Ex: Blocks World

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

**Output:**
Plan Space

- Need less abstract / better motivated example

At-home
- Drive
- Uber
- LINK
  - At-SEATAC
  - At-SEATAC
  - At-SEATAC

Plan Space

Visit FEZ
- Add Action
- Camel Ride
- Constrain Ordering

Camel Ride < Visit FEZ
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 modificat’n 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 operators are represented by arcs
  - Edges may be labeled with costs
  - We can rarely build this graph in memory (so we don’t)

Ridiculously tiny search graph for a tiny search problem

State Space Sizes?

- Search Problem: Eat all of the food
- Pacman positions: \(10 \times 12 = 120\)
- Pacman facing: up, down, left, right
- Food configurations: \(2^{30}\)
- Ghost1 positions: 12
- Ghost 2 positions: 11

\[120 \times 4 \times 2^{30} \times 12 \times 11 = 6.8 \times 10^{13}\]
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 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

- Vertices in state space graphs are problem states
  - Represent an abstracted state of the world
  - Have successors, can be goal / non-goal, have multiple predecessors
- Vertices in search trees (“Nodes”) are plans
  - Contain a problem state and one parent, a path length, a depth & a cost
  - Represent a plan (sequence of actions) which results in the node’s state
  - The same problem state may be achieved by multiple search tree nodes

Building Search Trees

- Search:
  - Expand out possible nodes (plans) in the tree
  - Maintain a fringe of unexpanded nodes
  - Try to expand as few nodes as possible
General Tree Search

```
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
```

Important ideas:
- Fringe (leaves of tree)
- Expansion (adding successors of a leaf)
- 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 stack - LIFO
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 queue - FIFO
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>Definition</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>
Infinite paths make DFS incomplete...
- How can we fix this?
- Check new nodes against path from S
- Infinite search spaces still a problem

### 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 w/ Path Checking

- 1 node
- b nodes
- b^2 nodes
- b^m nodes

<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 if finite</td>
<td>N</td>
<td>O(b^m)</td>
<td>O(bm)</td>
</tr>
</tbody>
</table>

* Or graph search – next lecture.
When is BFS optimal?

<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>(N) unless finite</td>
<td>(N)</td>
<td>(O(b^m))</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>
</tbody>
</table>

Memory a Limitation?

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