Problem Spaces & Search

- CSE 573

Logistics

- Mailing list
- Reading
  - Ch 4.2, Ch 6
- Mini-Project I
  - Partners
- Game Playing?

573 Topics

- Multi-agent
- NLP
- Robotics
- Softbots

<table>
<thead>
<tr>
<th>Reinforcement Learning</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference</td>
<td></td>
</tr>
<tr>
<td>Logic-Based</td>
<td>Knowledge Representation</td>
</tr>
<tr>
<td>Supervised Learning</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>Search</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>Problem Spaces</td>
<td>Planning</td>
</tr>
<tr>
<td>Agency</td>
<td></td>
</tr>
</tbody>
</table>

Weak Methods

- "In the knowledge lies the power..."
  - [Feigenbaum]
- But what if you have very little knowledge???

Generate & Test

- As weak as it gets...
- Works on semi-decidable problems!

Mass Spectrometry

\[ \text{C}_2\text{O}_4\text{N}_2\text{H}_2\text{O} \]

Relative abundance vs. m/z value
**Search thru a Problem Space / State Space**

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

- **Output:**
  - Path: start ⇒ a state satisfying goal test
  - [May require shortest path]

---

**Example: Route Planning**

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

- **Output:**

---

**Example: N Queens**

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

---

**Multiple Problem Spaces**

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

- **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
**Classifying Search**

- **GUESSING ("Tree Search")**
  - Guess how to extend a partial solution to a problem.
  - Generates a tree of (partial) solutions.
  - The leaves of the tree are either "failures" or represent complete solutions.

- **SIMPPLYING ("Inference")**
  - Infer new, stronger constraints by combining one or more constraints (without any "guessing").
  - Example: \( X+2Y = 3 \)
  - \( X+Y = 1 \)
  - Therefore \( Y = 2 \)

- **WANDERING ("Markov chain")**
  - Perform a (biased) random walk through the space of (partial or total) solutions.

---

**Search Strategies v2**

- **Blind Search**
  - Depth first search
  - Breadth first search
  - Iterative deepening search
  - Iterative broadening search

- **Informed Search**
  - Constraint Satisfaction
  - Adversary Search

---

**Depth First Search**

- Maintain stack of nodes to visit
- Evaluation
  - Complete?
  - Not for infinite spaces
- Time Complexity?
  - \( O(b^d) \)
- Space Complexity?
  - \( O(bd) \)

---

**Breadth First Search**

- Maintain queue of nodes to visit
- Evaluation
  - Complete?
    - Yes (if branching factor is finite)
  - Time Complexity?
    - \( O(b^{d+1}) \)
  - Space Complexity?
    - \( O(b^{d+1}) \)

---

**Memory a Limitation?**

- **Suppose:**
  - 2 GHz CPU
  - 1 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node
  - 200,000 expansions / sec
  - Memory filled in 100 sec ... < 2 minutes
Iterative Deepening Search

- DFS with limit: incrementally grow limit
- Evaluation
  - Complete? Yes
  - Time Complexity? $O(b^d)$
  - Space Complexity? $O(d)$

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>

Speed

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>Iter. Deep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Time</td>
<td>Nodes</td>
</tr>
<tr>
<td>8 Puzzle</td>
<td>$10^5$</td>
<td>.01 sec</td>
</tr>
<tr>
<td>2x2x2 Rubik’s</td>
<td>$10^6$</td>
<td>.2 sec</td>
</tr>
<tr>
<td>15 Puzzle</td>
<td>$10^{13}$</td>
<td>6 days</td>
</tr>
<tr>
<td>3x3x3 Rubik’s</td>
<td>$10^{19}$</td>
<td>68k yrs</td>
</tr>
<tr>
<td>24 Puzzle</td>
<td>$10^{25}$</td>
<td>128 yrs</td>
</tr>
</tbody>
</table>

When to Use Iterative Deepening

- N Queens?

Search Space with Uniform Structure

Search Space with Clustered Structure
Iterative Broadening Search

- What if know solutions lay at depth N?
- No sense in doing iterative deepening
- Increment sum of sibling indices

Forwards vs. Backwards

Problem

- All these methods are slow (blind)
- Solution
  → add guidance ("heuristic estimate")
  → "informed search"
  ... coming next ...

Recap: Search thru a Problem Space / State Space

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]
- Output:
  - Path: start ⇒ a state satisfying goal test
  - [May require shortest path]

Cryptarithmetic

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:
  - SEND
  - MORE
  - MONEY
Concept Learning

Labeled Training Examples

\[
\begin{align*}
&<p1, \text{blond}, \text{32}, \text{mc}, \text{ok}> \\
&<p2, \text{red}, \text{47}, \text{visa}, \text{ok}> \\
&<p3, \text{blond}, \text{23}, \text{cash}, \text{ter}> \\
&\vdots
\end{align*}
\]

Output: \( f: <\text{pn}...> \rightarrow \{\text{ok, ter}\} \)

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

Symbolic Integration

• E.g. \( \int x^2e^x \, dx = e^x(x^2-2x+2) + C \)

Operators:
  • Integration by parts
  • Integration by substitution
  ...

Search Strategies v2

• Blind Search
• Informed Search
  • Best-first search
  • \( A^* \) search
  • Iterative deepening \( A^* \) search
  • Local search
• Constraint Satisfaction
• Adversary Search

Best-first 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

Old Friends

• Breadth first = best first
  • With \( f(n) = \text{depth}(n) \)
• Dijkstra's Algorithm = best first
  • With \( f(n) = g(n) \), i.e. the sum of edge costs from start to \( n \)
  • Space bound (stores all generated nodes)

\( A^* \) Search

• Hart, Nilsson & Rafael 1968
• Best first search with \( f(n) = g(n) + h(n) \)
• Where \( g(n) = \text{sum of costs from start to } n \)
• \( h(n) = \text{estimate of lowest cost path } n \rightarrow \text{goal} \)
• \( h(\text{goal}) = 0 \)
• If \( h(n) \) is admissible (and monotonic) then \( A^* \) is optimal
  • Underestimates cost of any solution which can reached from node
  • \( f \) values increase from node to descendant's (triangle inequality)
A* Example

Optimality of A*

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

\[ f(G_2) = g(G_2) \quad \text{since} \quad h(G_2) = 0 \]
\[ > g(G_1) \quad \text{since} \quad G_2 \text{ is suboptimal} \]
\[ \geq f(n) \quad \text{since} \quad h \text{ is admissible} \]

Since $f(G_2) > f(n)$, $A^*$ will never select $G_2$ for expansion.

Optimality Continued

Lemma: $A^*$ expands nodes in order of increasing $f$ value

Gradually adds "$f$-contours" of nodes (cf. breadth-first adds layers)

Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$

A* Summary

Pros

Cons

Iterative-Deepening A*

Like iterative-deepening depth-first, but...

- Depth bound modified to be an $f$-limit
- Start with limit = $h$(start)
- Prune any node if $f$(node) > $f$-limit
- Next $f$-limit = min-cost of any node pruned

IDA* Analysis

- Complete & Optimal (ala $A^*$)
- Space usage $\propto$ depth of solution
- Each iteration is DFS - no priority queue!
- # nodes expanded relative to $A^*$
  - Depends on # unique values of heuristic function
  - In 8 puzzle: few values $\Rightarrow$ close to # $A^*$ expands
  - In traveling salesman: each $f$ value is unique
    $\Rightarrow 1 + 2 + \ldots + n = O(n^2)$ where $n$ nodes $A^*$ expands
    if $n$ is too big for main memory, $n^2$ is too long to wait!
- Generates duplicate nodes in cyclic graphs