Analysis of Uninformed Search Methods

Brian C. Williams
16.410
Feb 18th, 2003

Slides adapted from:
6.034 Tomas Lozano Perez,
Russell and Norvig AIMA
Assignment

• Reading:
  – Solving problems by searching: AIMA Ch. 3
  – Informed search and exploration: AIMA Ch. 4.1-2

• Homework:
  – Online problem set 2 due next Monday Feb 24\textsuperscript{th}
Outline

• Recap

• Analysis
  – Depth-first search
  – Breadth-first search

• Iterative deepening
Complex missions must carefully:

- Plan complex sequences of actions
- Schedule tight resources
- Monitor and diagnose behavior
- Repair or reconfigure hardware.

⇒ Most AI problems, like these, may be formulated as state space search.
Problem Solving Searches
Paths in a Graph

• Formulate Goal
• Formulate Problem
  – States
  – Operators
• Generate Solution
  – Sequence of states
Depth First Search (DFS)

Depth-first:
- Add path extensions to front of Q
- Pick first element of Q

Breadth First Search (BFS)

Breadth-first:
- Add path extensions to back of Q
- Pick first element of Q
Simple Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S) as only entry; set Visited = ()
2. If Q is empty, fail. Else, pick some partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else
   a) Remove N from Q
   b) Find all children of head(N) not in Visited and create all the one-step extensions of N to each child.
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited
   e) Go to step 2.
Outline

• Recap
• **Analysis**
  – Depth-first search
  – Breadth-first search
• Iterative deepening
Elements of Algorithmic Analysis

- **Soundness:**
  - is a solution returned by the algorithm guaranteed to be correct?

- **Completeness:**
  - is the algorithm guaranteed to find a solution when there is one?

- **Optimality:**
  - is the algorithm guaranteed to find a best solution when there is one?

- **Time complexity:**
  - how long does it take to find a solution?

- **Space complexity:**
  - how much memory does it need to perform search?
Characterizing Search Algorithms

Level 0
Level 1
Level 2

b = maximum branching factor, number of children

b = 3

m = maximum length of any path in the state space

m = 1

d = depth of the shallowest goal node

d = 1
Cost and Performance

Which is better, *depth-first* or *breadth-first*?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breadth-first</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
Worst Case Time for Depth-first

Worst case time $T$ is proportional to number of nodes visited

Level 0

Level 1

Level 2

Level $m$

$$T_{dfs} = b^m + \ldots b + 1$$

$$b \cdot T_{dfs} = b^{m+1} + b^m + \ldots b$$

$$[b - 1] \cdot T_{dfs} = b^{m+1} - 1$$

Solve recurrence

$$T_{dfs} = \frac{[b^{m+1} - 1]}{[b - 1]} \cdot c_{dfs}$$

where $c_{dfs}$ is time per node
Cost Using Order Notation

Worst case time \( T \) is proportional to number of nodes visited

Order Notation
- \( T = O(e) \) if \( T = c \times e \) for some constant \( c \)

\[ T_{dfs} = [b^m + \ldots + b + 1] \times c_{dfs} \]

\[ = O(b^m) \text{ for large } b \]
\[ = O(b^{m+1}) \text{ more conservatively} \]
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>$b^m$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breadth-first</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
Worst Case Space for Depth-first

Worst case space \( S_{dfs} \) is proportional to maximal length of Q

- If a node is queued its parent and parent’s siblings have been queued.
  \( S_{dfs} = (b-1) \cdot m + 1 \)

The children of at most one sibling is expanded at each level.
  \( S_{dfs} = (b-1) \cdot m + 1 \)
  
- \( S_{dfs} = O(b \cdot m) \)
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>$b^m$</td>
<td>$b^*m$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breadth-first</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>$b^m$</td>
<td>$b^*m$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Breadth-first</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of $Q$
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>$b^m$</td>
<td>$b^*m$</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of $Q$
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>(b^m)</td>
<td>(b^*m)</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
Worst Case Time for Breadth-first

Worst case time \( T \) is proportional to number of nodes visited

\[
T_{\text{bfs}} = \left[ b^{d+1} + b^d + \ldots + b + 1 - b \right] \cdot c_{\text{bfs}} = O(b^{d+1})
\]
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>$b^m$</td>
<td>$b^*m$</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>$b^{d+1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of $Q$
Worst Case Space for Breadth-first

Worst case space $S_{\text{dfs}}$ is proportional to maximal length of $Q$

$$S_{\text{bfs}} = [b^{d+1} - b + 1] \cdot c_{\text{bfs}}$$
$$= O(b^{d+1})$$
Cost and Performance

Which is better, depth-first or breadth-first?

![Graphs showing depth-first and breadth-first search]

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>(b^m)</td>
<td>(b^*m)</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>(b^{d+1})</td>
<td>(b^{d+1})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
Breadth-first Finds Shortest Path

Assuming each edge is length 1, other paths to G must be at least as long as first found

Nodes visited earlier can’t include G

First reached
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>(b^m)</td>
<td>(b^*m)</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>(b^{d+1})</td>
<td>(b^{d+1})</td>
<td><strong>Yes</strong></td>
<td></td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited

Worst case space is proportional to maximal length of Q
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>$B^m$</td>
<td>$b^*m$</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>$b^{d+1}$</td>
<td>$b^{d+1}$</td>
<td>Yes (unit length)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
### Growth for Best First Search

$b = 10$; $10,000$ nodes/sec; $1000$ bytes/node

<table>
<thead>
<tr>
<th>Depth</th>
<th>Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1,100</td>
<td>.11 seconds</td>
<td>1 megabyte</td>
</tr>
<tr>
<td>4</td>
<td>111,100</td>
<td>11 seconds</td>
<td>106 megabytes</td>
</tr>
<tr>
<td>6</td>
<td>$10^7$</td>
<td>19 minutes</td>
<td>10 gigabytes</td>
</tr>
<tr>
<td>8</td>
<td>$10^9$</td>
<td>31 hours</td>
<td>1 terabyte</td>
</tr>
<tr>
<td>10</td>
<td>$10^{11}$</td>
<td>129 days</td>
<td>101 terabytes</td>
</tr>
<tr>
<td>12</td>
<td>$10^{13}$</td>
<td>35 years</td>
<td>10 petabytes</td>
</tr>
<tr>
<td>14</td>
<td>$10^{15}$</td>
<td>3,523 years</td>
<td>1 exabyte</td>
</tr>
</tbody>
</table>
Cost and Performance

Which is better, depth-first or breadth-first?

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>$b^n$</td>
<td>$b^*m$</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>$b^{d+1}$</td>
<td>$b^{d+1}$</td>
<td>Yes unit length</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
Cost and Performance 6.034 Style: What the Electronic Tutor Thinks
Which is better, depth-first or breadth-first?

- Assumes \( d = m \) in the worst case, and calls both \( d \).
- Takes the conservative estimate: \( b^d + \ldots + 1 = O(b^{d+1}) \)

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Worst Time</th>
<th>Worst Space</th>
<th>Shortest Path?</th>
<th>Guaranteed to find path?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>( b^{d+1} )</td>
<td>( b^d )</td>
<td>No</td>
<td>Yes for finite graph</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>( b^{d+1} )</td>
<td>( b^d )</td>
<td>Yes ( \text{unit length} )</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q
Outline

• Recap
• Analysis
• Iterative deepening
Iterative Deepening (IDS)

Idea:
• Explore tree in breadth-first order, using depth-first search.
→ Search tree to depth 1, ....
Iterative Deepening (IDS)

Idea:
• Explore tree in breadth-first order, using depth-first search.
  ➔ Search tree to depth 1, then 2, ....
Iterative Deepening (IDS)

Idea:
- Explore tree in breadth-first order, using depth-first search.
  ➔ Search tree to depth 1, then 2, then 3….
Cost of Iterative Deepening

\[ T_{ids} = d + 1 + (d)b + (d - 1)b^2 + \ldots b^d = O(b^d) \]

\[ T_{bfs} = 1 + b + b^2 + \ldots b^d + (b^{d+1} - b) = O(b^{d+1}) \]

Iterative deepening performs better than breadth-first!
Summary

• Most problem solving tasks may be encoded as state space search.
• Basic data structures for search are graphs and search trees.
• Depth-first and breadth-first search may be framed, among others, as instances of a generic search strategy.
• Cycle detection is required to achieve efficiency and completeness.
• Complexity analysis shows that breadth-first is preferred in terms of optimality and time, while depth-first is preferred in terms of space.
• Iterative deepening draws the best from depth-first and breadth-first search.