Basic Search

CPSC 470 – Artificial Intelligence
Brian Scassellati
## Characterizing Sample Environments

<table>
<thead>
<tr>
<th>Environment</th>
<th>Observable</th>
<th>Deterministic</th>
<th>Episodic</th>
<th>Static</th>
<th>Discrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess (no clock)</td>
<td>Fully</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Poker</td>
<td>Partially</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Taxi driving</td>
<td>Partially</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Image analysis</td>
<td>Fully</td>
<td>Yes</td>
<td>Yes</td>
<td>Semi</td>
<td>No</td>
</tr>
<tr>
<td>Part-picking robot</td>
<td>Partially</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Problem Formulation

- Well-defined function that identifies both the goal states and the conditions under which to achieve the goal
  - Fly from Boston to San Francisco
  - Quality might depend on
    - Least amount of money
    - Fewest number of transfers
    - Shortest amount of time in the air
    - Shortest amount of time in airports
Problem Formulation

• Well-defined problems
  – Fully observable
  – Deterministic
  – Static
  – Discrete set of possible actions (operations)

• **State space**: the set of all states that are reachable from an initial state by any sequence of actions

• **Path**: sequence of actions leading from one state to another
Problem Formulation

- Goal: spend less $
- State space: flights and their costs
- Path: sequence of flights
- Picking the right level of abstraction
  - Fly from Boston to Chicago
  - Directions to the airport
  - Move left leg 18 inches forward
Problem Formulation Matters!
The 8 Queens Problem

• Formulation #1:
  – Place a queen on any open square
  – Repeat until all queens are placed
  – State space of
    $64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57 = 1.78 \times 10^{14}$

• Formulation #2:
  – Place a queen on any square in row 1
  – Place a queen on any square in row 2
  – State space of
    $8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 = 1.68 \times 10^{7}$

• Formulation #3:
  – Place a queen on any square in row 1
  – Place a queen on a square in row 2 that is not in the same column…
  – State space of
    $8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 40,320$
Problem Formation involves Abstraction: Missionaries and Cannibals

- 3 missionaries and 3 cannibals on left side
- Boat holds 1 or 2 people
- Never leave missionaries outnumbered by cannibals
- States:
  - (# cannibals, # missionaries, # boats) on left side of river
- Operators
  - Remove up to 2 people to other side
Real-World Applications: VLSI Layout

(Images from Cadence Inc.’s Virtuoso System)
Real-World Applications: Traveling Salesman Problem
How to Search:
Generating Sequences and Data Structures

Branching Factor $b=3$
Measuring Performance

• **Completeness**: is the strategy guaranteed to find a solution when one exists?
• **Time Complexity**: how long does it take to find a solution?
• **Space Complexity**: how much memory does it require to perform the search?
• **Optimality**: Does the strategy find the best-quality solution when more than one solution exists?
Breadth-First Search

- Finds the most shallow solution
- Complete
- Optimal when the path cost is a non-decreasing function of depth

<table>
<thead>
<tr>
<th>Depth</th>
<th>Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 millisecond</td>
<td>100 bytes</td>
</tr>
<tr>
<td>2</td>
<td>111</td>
<td>.1 seconds</td>
<td>11 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>11,111</td>
<td>11 seconds</td>
<td>1 megabyte</td>
</tr>
<tr>
<td>6</td>
<td>$10^6$</td>
<td>18 minutes</td>
<td>111 megabytes</td>
</tr>
<tr>
<td>8</td>
<td>$10^8$</td>
<td>31 hours</td>
<td>11 gigabytes</td>
</tr>
<tr>
<td>10</td>
<td>$10^{10}$</td>
<td>128 days</td>
<td>1 terabyte</td>
</tr>
<tr>
<td>12</td>
<td>$10^{12}$</td>
<td>35 years</td>
<td>111 terabytes</td>
</tr>
<tr>
<td>14</td>
<td>$10^{14}$</td>
<td>3500 years</td>
<td>11,111 terabytes</td>
</tr>
</tbody>
</table>

- Assuming
  - Branching factor $b=10$
  - Process 1000 nodes/sec
  - 100 bytes/node
- Time is a big issue
- Space is a bigger issue
- Exponential growth leads to impractical problems for uninformed search
Uniform Cost Search

- Travel from the start (S) to the goal (G)
- Cost associated with each link
- Always expand the fringe node with the lowest cost
- Breadth-first search is uniform search with cost=depth
Depth-First Search

- Minimal memory requirements (only stores one path at a time)
- Best case scenario
- Worst case scenario
- Non-optimal
- What happens on trees with infinite depth?
  - Completeness is sacrificed
Depth Limited

- Follow depth-first search, but with a maximum depth
- Requires some knowledge of the solution:
  - 9 cities, depth limit of 8?
- Non-optimal
- What if we choose a limit too small?
  - Sacrifice completeness
Iterative Deepening

- Tries all possible depth limits (l=0,1,2,...)
- Cost of re-computing the lower depths
  - But most nodes are in the deep bottom of the tree
  - Tree with depth 3, branching factor 2
    - $1+2+4+8 = 15$ nodes for pure depth first search
    - $3+7+15 = 25$ nodes for iterative deepening search
  - Tree with depth 5, branching factor 10
    - $1+10+100+1,000+10,000+100,000 = 111,111$ nodes depth-first
    - $6+50+400+3,000+20,000+100,000 = 123,456$ nodes iterative depth
Bidirectional Search

• If you can work backward from the solution, then you can limit the search depth
• With a solution at depth \(d\), then find a solution in \(O(2b^{d/2}) = O(b^{d/2})\) steps
• Better than \(O(b^d)\) steps with breadth-first search!
  – For a tree with \(b=10\), \(d=6\)
    • Breadth-first search generates 1,111,111 nodes
    • Bi-directional search generates 2,222 nodes
Comparison of Techniques

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>( b^d )</td>
<td>( b^d )</td>
<td>( b^m )</td>
<td>( b^l )</td>
<td>( b^d )</td>
<td>( b^{d/2} )</td>
</tr>
<tr>
<td>Space</td>
<td>( b^d )</td>
<td>( b^d )</td>
<td>( bm )</td>
<td>( bl )</td>
<td>( bd )</td>
<td>( b^{d/2} )</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if ( l \geq d )</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- \( b \) = branching factor
- \( d \) = depth of solution
- \( m \) = maximum depth of tree
- \( l \) = depth limit
Still not as smart as it could be...
Coming Up Next

• More intelligent search strategies
  – Best-first search
  – A* search
  – Heuristic search

• Applications
  – Playing games
  – Constraint satisfaction problems
Administrivia

- Office hours posted today
- PS 0 due today at 11:59pm
- PS 1 out today… search in PACMAN
Sign Up to Work on a Collaborative Task with a Robot

And earn $10!

Sign up at: tinyurl.com/yale-robot-team-task