Problem Solving

Russell and Norvig: Chapter 3

CSMSC 421 – Fall 2003

Announcements and Outline

- Announcements:
  - Lise’s office hours: Thu 1:30-3:00 and by appt
  - HW1 due next Tue

- Outline:
  - Problem Solving Agents
  - Problem Formulation
  - Basic Search

Problem-Solving Agent

- Formulate Goal
- Formulate Problem
- States
- Actions
- Find Solution
Example: Route finding

- On holiday in Romania; Currently in Arad.
- Flight leaves tomorrow from Bucharest.
- Formulate Goal:
  - Be in Bucharest
- Formulate Problem:
  - States: various cities
  - Actions: drive between cities
- Find solution:
  - Sequence of cities: Arad, Sibiu, Fagaras, Bucharest

Problem Formulation

Vacuum World
Search Problem

- State space
  - each state is an abstract representation of the environment
  - the state space is discrete
- Initial state
- Successor function
- Goal test
- Path cost

Search Problem

- State space
- Initial state:
  - usually the current state
  - sometimes one or several hypothetical states ("what if ...")
- Successor function
- Goal test
- Path cost

Search Problem

- State space
- Initial state
- Successor function:
  - [state → subset of states]
  - an abstract representation of the possible actions
- Goal test
- Path cost

Search Problem

- State space
- Initial state
- Successor function
- Goal test:
  - usually a condition
  - sometimes the description of a state
- Path cost
Search Problem

- State space
- Initial state
- Successor function
- Goal test
- Path cost:
  - [path → positive number]
  - usually, path cost = sum of step costs
  - e.g., number of moves of the empty tile

Example: 8-puzzle

Size of the state space = 9!/2 = 181,440

15-puzzle → \(0.65 \times 10^{12}\)

24-puzzle → \(0.5 \times 10^{25}\)

10 millions states/sec

0.18 sec

6 days

12 billion years
Your Turn: Search Problem

- State space
- Initial state
- Successor function
- Goal test
- Path cost

Search of State Space

Search State Space
Search of State Space

Simple Agent Algorithm

Problem-Solving-Agent
1. initial-state ← sense/read state
2. goal ← select/read goal
3. successor ← select/read action models
4. problem ← (initial-state, goal, successor)
5. solution ← search(problem)
6. perform(solution)
Example: 8-queens

Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

Formulation #1:
- States: any arrangement of 0 to 8 queens on the board
- Initial state: 0 queens on the board
- Successor function: add a queen in any square
- Goal test: 8 queens on the board, none attacked

→ $64^8$ states with 8 queens

Example: 8-queens

Formulation #2:
- States: any arrangement of $k = 0$ to 8 queens in the $k$ leftmost columns with none attacked
- Initial state: 0 queens on the board
- Successor function: add a queen to any square in the leftmost empty column such that it is not attacked by any other queen
- Goal test: 8 queens on the board

→ 2,067 states

Example: Robot navigation

What is the state space?
Example: Robot navigation

Cost of one horizontal/vertical step = 1
Cost of one diagonal step = \sqrt{2}

Example: Assembly Planning

Initial state

Complex function: it must find if a collision-free merging motion exists

Successor function:
• Merge two subassemblies

Goal state
**Example: Assembly Planning**

**Assumptions in Basic Search**
- The environment is static
- The environment is discretizable
- The environment is observable
- The actions are deterministic

**Simple Agent Algorithm**

Problem-Solving-Agent
1. initial-state $\leftarrow$ sense/read state
2. goal $\leftarrow$ select/read goal
3. successor $\leftarrow$ select/read action models
4. problem $\leftarrow$ (initial-state, goal, successor)
5. solution $\leftarrow$ search(problem)
6. perform(solution)

**Search of State Space**

$\Rightarrow$ search tree
Basic Search Concepts
- Search tree
- Search node
- Node expansion
- Search strategy: At each stage it determines which node to expand

Search Nodes ≠ States
- The search tree may be infinite even when the state space is finite

Fringe
- Set of search nodes that have not been expanded yet
- Implemented as a queue FRINGE
  - INSERT(node,FRINGE)
  - REMOVE(FRINGE)
- The ordering of the nodes in FRINGE defines the search strategy

Search Algorithm
1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,FRINGE)
3. Repeat:
   - If FRINGE is empty then return failure
   - n ← REMOVE(FRINGE)
   - s ← STATE(n)
   - For every state s' in SUCCESSORS(s)
     - Create a node n'
     - If GOAL?(s') then return path or goal state
     - INSERT(n',FRINGE)
Search Strategies

- A strategy is defined by picking the order of node expansion.
- Strategies are evaluated along the following dimensions:
  - Completeness – does it always find a solution if one exists?
  - Time complexity – number of nodes generated/expanded
  - Space complexity – maximum number of nodes in memory
  - Optimality – does it always find a least-cost solution
- Time and space complexity are measured in terms of:
  - $b$ – maximum branching factor of the search tree
  - $d$ – depth of the least-cost solution
  - $m$ – maximum depth of the state space (may be $\infty$)

Important Remark

- Some problems formulated as search problems are NP-hard problems. We cannot expect to solve such a problem in less than exponential time in the worst-case.
- But we can nevertheless strive to solve as many instances of the problem as possible.

Blind vs. Heuristic Strategies

- **Blind** (or uninformed) strategies do not exploit any of the information contained in a state.
- **Heuristic** (or informed) strategies exploit such information to assess that one node is "more promising" than another.

Blind Search...

- "[the ant] knew that a certain arrangement had to be made, but it could not figure out how to make it. It was like a man with a tea-cup in one hand and a sandwich in the other, who wants to light a cigarette with a match. But, where the man would invent the idea of putting down the cup and sandwich—before picking up the cigarette and the match—this ant would have put down the sandwich and picked up the match, then it would have been down with the match and up with the cigarette, then down with the cigarette and up with the sandwich, then down with the cup and up with the cigarette, until finally it had put down the sandwich and picked up the match. It was inclined to rely on a series of accidents to achieve its object. It was patient and did not think…" Wart watched the arrangements with a surprise which turned into vexation and then into dislike. He felt like asking why it did not think things out in advance…
  
T.H. White, *The Once and Future King*
**Blind Strategies**

- **Breadth-first**
  - Bidirectional
- **Depth-first**
  - Depth-limited
  - Iterative deepening
- **Uniform-Cost**
  - Step cost = \( c \) \( \geq \varepsilon > 0 \)

**Breadth-First Strategy**

New nodes are inserted at the end of FRINGE

FRINGE = (1)

FRINGE = (2, 3)

FRINGE = (3, 4, 5)
**Breadth-First Strategy**

New nodes are inserted at the end of FRINGE

FRINGE = (4, 5, 6, 7)

---

**Evaluation**

- \( b \): branching factor
- \( d \): depth of shallowest goal node
- Complete
- Optimal if step cost is 1

Number of nodes generated:

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

Time and space complexity is \( O(b^{d+1}) \)

---

**Time and Memory Requirements**

<table>
<thead>
<tr>
<th>( d )</th>
<th>#Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>111</td>
<td>.01 msec</td>
<td>11 Kbytes</td>
</tr>
<tr>
<td>4</td>
<td>11,111</td>
<td>1 msec</td>
<td>1 Mbyte</td>
</tr>
<tr>
<td>6</td>
<td>( \sim 10^6 )</td>
<td>1 sec</td>
<td>100 Mb</td>
</tr>
<tr>
<td>8</td>
<td>( \sim 10^8 )</td>
<td>100 sec</td>
<td>10 Gbytes</td>
</tr>
<tr>
<td>10</td>
<td>( \sim 10^{10} )</td>
<td>2.8 hours</td>
<td>1 Tbyte</td>
</tr>
<tr>
<td>12</td>
<td>( \sim 10^{12} )</td>
<td>11.6 days</td>
<td>100 Tbytes</td>
</tr>
<tr>
<td>14</td>
<td>( \sim 10^{14} )</td>
<td>3.2 years</td>
<td>10,000 Tb</td>
</tr>
</tbody>
</table>

Assumptions: \( b = 10 \); 1,000,000 nodes/sec; 100 bytes/node
**Bidirectional Strategy**

2 fringe queues: FRINGE1 and FRINGE2

Time and space complexity $= O(b^{d/2}) << O(b^d)$

**Depth-First Strategy**

New nodes are inserted at the front of FRINGE

FRINGE = (1)

FRINGE = (2, 3)

FRINGE = (4, 5, 3)
Depth-First Strategy

New nodes are inserted at the front of FRINGE

Depth-First Strategy

New nodes are inserted at the front of FRINGE

Depth-First Strategy

New nodes are inserted at the front of FRINGE

Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy
New nodes are inserted at the front of FRINGE

Depth-First Strategy
New nodes are inserted at the front of FRINGE
**Evaluation**
- **b**: branching factor
- **d**: depth of shallowest goal node
- **m**: maximal depth of a leaf node
- Complete only for finite search tree
- Not optimal
- Number of nodes generated: 
  \[1 + b + b^2 + \ldots + b^m = O(b^m)\]
- Time complexity is \(O(b^m)\)
- Space complexity is \(O(bm)\)

**Depth-Limited Strategy**
- Depth-first with **depth cutoff** \(k\) (maximal depth below which nodes are not expanded)
- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

**Iterative Deepening Strategy**
- Repeat for \(k = 0, 1, 2, \ldots\):
  - Perform depth-first with depth cutoff \(k\)
- Complete
- Optimal if step cost =1
- Time complexity is:
  \[(d+1)(1) + db + (d-1)b^2 + \ldots + (1)b^d = O(b^d)\]
- Space complexity is: \(O(bd)\)

**Comparison of Strategies**
- Breadth-first is complete and optimal, but has high space complexity
- Depth-first is space efficient, but neither complete nor optimal
- Iterative deepening is asymptotically optimal
Repeated States

Avoiding Repeated States

Avoiding Repeated States

Detecting Identical States

Search tree is finite

Search tree is infinite

Avoiding Repeated States

Depth-first strategy:

- Solution 1:
  - Keep track of all states associated with nodes in current tree
  - If the state of a new node already exists, then discard the node
  - Avoids loops

- Solution 2:
  - Keep track of all states generated so far
  - If the state of a new node has already been generated, then discard the node
  - Space complexity of breadth-first

Requires comparing state descriptions

Breadth-first strategy:

- Keep track of all generated states
- If the state of a new node already exists, then discard the node

Detecting Identical States

- Use explicit representation of state space
- Use hash-code or similar representation
Uniform-Cost Strategy

- Each step has some cost \( \geq \varepsilon > 0 \).
- The cost of the path to each fringe node \( N \) is \( g(N) = \Sigma \) costs of all steps.
- The goal is to generate a solution path of minimal cost.
- The queue FRINGE is sorted in increasing cost.

Modified Search Algorithm

1. INSERT(initial-node,FRINGE)
2. Repeat:
   - If FRINGE is empty then return failure
   - \( n \leftarrow \text{REMOVE}(\text{FRINGE}) \)
   - \( s \leftarrow \text{STATE}(n) \)
   - If GOAL?(s) then return path or goal state
   - For every state \( s' \) in SUCCESSORS(s)
     - Create a node \( n' \)
     - INSERT(\( n', \text{FRINGE} \))

Exercises

- Adapt uniform-cost search to avoid repeated states while still finding the optimal solution
- Uniform-cost looks like breadth-first (it is exactly breadth first if the step cost is constant). Adapt iterative deepening in a similar way to handle variable step costs

Summary

- Search tree \( \neq \) state space
- Search strategies: breadth-first, depth-first, and variants
- Evaluation of strategies: completeness, optimality, time and space complexity
- Avoiding repeated states
- Optimal search with variable step costs