Search

EECS 348

Intro to Artificial Intelligence

(slides from Oren Etzioni, based on Stuart Russell, Dan Weld, Henry Kautz, and others)
What is Search?

Search is a class of techniques for systematically finding or constructing solutions to problems.

Example technique: generate-and-test.

Example problem: Combination lock.

1. Generate a possible solution.
2. Test the solution.
3. If solution found THEN done ELSE return to step 1.
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]
Why is search interesting?

Many (all?) AI problems can be formulated as search problems!

Examples:
- Path planning
- Games
- Natural Language Processing
- Machine learning
- Genetic algorithms
Example: The 8-puzzle

states?  
actions?  
goal_test?  
path_cost?
Example: The 8-puzzle

- **States**: locations of tiles
- **Actions**: move blank left, right, up, down
- **Goal Test**: = goal state (given)
- **Path Cost**: 1 per move

[Note: optimal solution of $n$-Puzzle family is NP-hard]
Search Tree Example:
Fragment of 8-Puzzle Problem Space
Example: robotic assembly

**states?**: real-valued coordinates of robot joint angles

**parts of the object to be assembled**

**actions?**: continuous motions of robot joints

**goal test?**: complete assembly

**path cost?**: time to execute
Implementation: states vs. nodes

A state is a (representation of) a physical configuration.

A node is a data structure constituting part of a search tree includes state, parent node, action, path cost $g(x)$, depth.

The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.
Tree Search

Frontier = root node
Repeat while Frontier non-empty
  Take n from Frontier
  For all n’ in Expand(n)
    if n’ is a goal, return it
    else add n’ to Frontier

Return failure
A search 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
- **space complexity**: maximum number of nodes in memory
- **optimality**: does it always find a least-cost solution?
- **systematicity**: does it visit each state at most once?

Time and space complexity are measured in terms of:

- **$b$**: maximum branching factor of the search tree
- **$d$**: depth of the shallowest solution
- **$m$**: maximum depth of the state space (may be $\infty$)
Uninformed search strategies use only the information available in the problem definition.

- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search
Depth First Search

Maintain stack of nodes to visit

Evaluation

• Complete?
  Not for infinite spaces
• Time Complexity?
  $O(b^m)$
• Space Complexity?
  $O(mb)$
  (with “backtracking” can be $O(m)$)
Breadth First Search

Maintain queue of nodes to visit

Evaluation

- **Complete?**
  Yes (assume $b$ finite)

- **Time Complexity?**
  $O(b^d)$

- **Space Complexity?**
  $O(b^d)$
BFS: Memory Limitation

Suppose:

- 4-core 2 GHz CPU
- 8 GB main memory
- 100 cycles / expansion
- 10 bytes / node

800,000 expansions / sec
Memory filled in 100 sec ... < 2 minutes
Iterative deepening search

function \textsc{iterative-deepening-search}( \textit{problem} ) \textbf{return}s a solution, or failure

\textbf{inputs:} \textit{problem}, a problem

\textbf{for} depth $\leftarrow 0$ \textbf{to} $\infty$ \textbf{do}

\hspace{1em} result $\leftarrow$ \textsc{depth-limited-search}( \textit{problem}, depth )

\hspace{1em} \textbf{if} result $\neq$ cutoff \textbf{then} \textbf{return} result
Iterative deepening search / =0

Limit = 0

Diagram showing a search process with a limit of 0.
Iterative deepening search / =1

Limit = 1

Diagram showing iterative deepening search with limit 1.
Iterative deepening search / =2

Limit = 2
Iterative deepening search / = 3
Iterative deepening search

Number of nodes generated in a depth-limited search to depth $d$ with branching factor $b$:

$$N_{DLS} = b^0 + b^1 + b^2 + ... + b^{d-2} + b^{d-1} + b^d$$

Number of nodes generated in an iterative deepening search to depth $d$ with branching factor $b$:

$$N_{IDS} = (d+1)b^0 + d b^1 + (d-1)b^2 + ... + 3b^{d-2} + 2b^{d-1} + 1b^d$$

For $b = 10$, $d = 5$,

- $N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111$
- $N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$

Overhead = $(123,456 - 111,111)/111,111 = 11\%$
# 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>
iterative deepening search

**Complete?** Yes

**Time?**
- \((d+1)b^0 + d b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)\)

**Space?**
- \(O(bd)\)

**Optimal?**
- Yes, if step cost = 1

**Systematic?**
- No, but okay
Repeated states (1 of 3)

Failure to detect repeated states can turn a linear problem into an exponential one!
Repeated states (2 of 3)

More realistic case:
Can save states but...then iterative deepening with DFS no longer takes $O(d)$ space!

What space is required?
Forwards vs. Backwards
vs. Bidirectional
Recap: Tree Search

Frontier = root node
Repeat while Frontier non-empty
    Take $n$ from Frontier
    For all $n'$ in Expand($n$)
        if $n'$ is a goal, return it
        else add $n'$ to Frontier

Return failure
Summary

Search problems

Search strategies

DFS, BFS, Iterative Deepening w/depth-limited DFS

Issue: All these methods are slow (blind)

Solution → add guidance ("heuristic estimate") → “informed search”
Survey!
Backup
Example: Romania

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, e.g., Arad, Sibiu, Fagaras, Bucharest
Example: N Queens

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

Output