Uninformed Search Strategies

AIMA 3.3-3.4

Review: Formulating search problems

- **Formulate search problem**
  - **States**: configurations of the puzzle (9! configurations)
  - **Actions**: Move one of the movable pieces (≤4 possible)
  - **Performance measure**: minimize total moves

- **Formulate goal**
  - Pieces to end up in order as shown...

- **Find solution**
  - Sequence of pieces moved: 3, 1, 6, 3, 1, ...

Review: Formulating search problems

- **States**: a set $S$
- **An initial state** $s_i \in S$
- **Actions**: a set $A$ — $∀ s \in S. \\text{Actions}(s) = \{ \text{actions that can be executed in } s \}$
- **Transition Model**: $∀ s \in S. ∀ a \in \text{Actions}(s). \\text{Result}(s, a) \rightarrow s_r$ — $s_r$ is called a successor of $s$
- **Path cost (Performance Measure)**: Must be additive
  - e.g. sum of distances, number of actions executed, ...
  - $c(s, a, y)$ is the step cost, assumed $\geq 0$
  - (where action $a$ goes from state $x$ to state $y$)
- **Goal test**: $\text{Goal}(s)$
  - Can be implicit, e.g. checkmate($s$)
  - $s$ is a goal state if $\text{Goal}(s)$ is true

Review: Solutions & Optimal Solutions

- A **solution** is a sequence of **actions** from the initial state to a **goal state**.

- **Optimal Solution**: A solution is optimal if no solution has a lower **path cost**.

Outline for today’s lecture

- **Search Fundamentals (AIMA 3.3)**
- Introduction to Uninformed Search
  - Review of Breadth first and Depth-first search
- Iterative deepening search
  - Strange Subroutine: Depth-limited search
  - Depth-limited search + iteration = WIN!!

Useful Concepts

- **State space**: the set of all states reachable from the initial state by any sequence of actions
  - When several operators can apply to each state, this gets large very quickly
  - Might be a proper subset of the set of configurations
- **Path**: a sequence of actions leading from one state $s_i$ to another state $s_f$
- **Frontier**: those states that are available for expanding (for applying legal actions to)
- **Solution**: a path from the initial state $s_i$ to a state $s_f$ that satisfies the goal test
Basic search algorithms: **Tree Search**

- Generalized algorithm to solve search problems (Review from Data Structures)
  - Enumerate in some order all possible paths from the initial state
  - Here: search through explicit tree generation
    - ROOT= initial state.
    - Nodes in search tree generated through transition model

Review: Generalized tree search

```plaintext
function TREE-SEARCH(problem, strategy) return a solution or failure
  Initialize frontier to the initial state of the problem
  do
    if the frontier is empty then return failure
    choose leaf node n for expansion according to strategy
    if n contains goal state then return solution
    else EXPAND n and add resulting nodes to the frontier
  end do
end function
```

8-Puzzle: States and Nodes

- A state is a (representation of a) **physical configuration**
- A node is a data structure constituting part of a search tree
  - Here node= <state, parent-node, children, action, path-cost, depth>
- States do not have parents, children, depth or path cost!
- The EXPAND function
  - uses the Actions and Transition Model to create the corresponding states
  - fills in the various fields of the new node data structure

Tree Search Flaw: Repeated states

- Failure to detect **repeated states** can turn a linear problem into an exponential one!

Solution: Graph Search!

- Simple Mod to tree search:
  - Check to see if a node has been visited before adding to search queue
    - must keep track of all possible states
    - can use a lot of memory
    - e.g., 8-puzzle problem, we have 9!/2 ≈ 182K states
Graph Search vs Tree Search

Outline for today's lecture

Search Fundamentals

Introduction to Uninformed Search (AIMA 3.4.1-3)
  • Review of Breadth-first and Depth-first search

Iterative deepening search
  • Strange Subroutine: Depth-limited search
  • Depth-limited search + iteration = WIN!!

Uninformed search strategies:

• AKA “Blind search”
• Uses only information available in problem definition

Informally:
  • Uninformed search: All non-goal nodes in frontier look equally good
  • Informed search: Some non-goal nodes can be ranked above others.

Search Strategies

• Review: Strategy = order of tree expansion
  • Implemented by different queue structures (LIFO, FIFO, priority)

Dimensions for evaluation
  • Completeness - always find the solution?
  • Optimality - finds a least cost solution (lowest path cost) first?
  • Time complexity - # of nodes generated (worst case)
  • Space complexity - # of nodes simultaneously in memory (worst case)

Time/space complexity variables
  • b, maximum branching factor of search tree
  • d, depth of the shallowest goal node
  • m, maximum length of any path in the state space (potentially ∞)

Introduction to space complexity

• You know about:
  • “Big O” notation
  • Time complexity

• Space complexity is analogous to time complexity

• Units of space are arbitrary
  • Doesn’t matter because Big O notation ignores constant multiplicative factors
  • Plausible Space units:
    — One Memory word
    — Size of any fixed size data structure
    — 4g Size of fixed size node in search tree

Review: Breadth-first search

• Idea:
  • Expand shallowest unexpanded node

• Implementation:
  • frontier is FIFO (First-In-First-Out) Queue:
    — Put successors at the end of frontier successor list.
Breadth-first search (simplified)

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
node = a node with STATE = problem.ROOT-STATE, PATH-COST=0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier = a FIFO queue with node as the only element
loop do
if EMPTY?(frontier) then return failure
node = POP(frontier) // chooses the shallowest node in frontier
add node. STATE to explored
for each action in problem.ACTIONS(node. STATE) do
child = CHILD-NODE(problem, node, action)
if problem.GOAL-TEST(child. STATE) then
return SOLUTION(child)
frontier = INSERT(child, frontier)

Properties of breadth-first search

- Complete? Yes (if b is finite)
- Time Complexity? $1+b+b^2+b^3+...+b^d = O(b^d)$
- Space Complexity? $O(b^d)$ (keeps every node in memory)
- Optimal? Yes, if cost = 1 per step (not optimal in general)

B: maximum branching factor of search tree
D: depth of the least cost solution
M: maximum depth of the state space ()

Exponential Space (and time) Not Good...

- Exponential complexity uninformed search problems cannot
  be solved for any but the smallest instances.
- (Memory requirements are a bigger problem than execution time.)

<table>
<thead>
<tr>
<th>DEPTH</th>
<th>NODES</th>
<th>TIME</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>110</td>
<td>0.11 ms</td>
<td>10⁷ B</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
<td>11 milliseconds</td>
<td>10⁶ Gb</td>
</tr>
<tr>
<td>6</td>
<td>10⁶</td>
<td>1.1 seconds</td>
<td>1 Gb</td>
</tr>
<tr>
<td>8</td>
<td>10⁸</td>
<td>2 minutes</td>
<td>10² Gb</td>
</tr>
<tr>
<td>10</td>
<td>10¹⁰</td>
<td>3 hours</td>
<td>10³ Gb</td>
</tr>
<tr>
<td>12</td>
<td>10¹²</td>
<td>13 days</td>
<td>10⁴ Gb</td>
</tr>
<tr>
<td>14</td>
<td>10¹⁴</td>
<td>3.5 years</td>
<td>10⁵ Gb</td>
</tr>
</tbody>
</table>

Fig 3.13 Assumes b=10, 1M nodes/sec, 1000 bytes/node

Depth-first vs Breadth-first

- Use depth-first if
  - Space is restricted
  - There are many possible solutions with long paths and wrong paths are usually terminated quickly
  - Search can be fine-tuned quickly

- Use breadth-first if
  - Possible infinite paths
  - Some solutions have short paths
  - Can quickly discard unlikely paths
Outline for today's lecture

- Formulating Search Problems – An Example
- Search Fundamentals
- Introduction to Uninformed Search
  - Review of Breadth first and Depth-first search
- **Iterative deepening search (AIMA 3.4.4-5)**
  - Strange Subroutine: Depth-limited search
  - Depth-limited search + iteration = WIN!!

Search Conundrum

- **Breadth-first**
  - Complete,
  - Optimal
  - but uses $O(b^d)$ space
- **Depth-first**
  - Not complete unless $m$ is bounded
  - Not optimal
  - Uses $O(b^m)$ time; terrible if $m >> d$
  - but only uses $O(b^m)$ space

How can we get the best of both?

Depth-limited search: A building block

- **Depth-First search but with depth limit $l$**
  - i.e. nodes at depth $l$ have no successors.
  - No infinite-path problem!

- If $l = d$ (by luck!), then optimal
  - But:
    - If $l < d$ then incomplete
    - If $l > d$ then not optimal

- Time complexity: $O(b^l)$
- Space complexity: $O(b^l)$

Iterative deepening search

- A general strategy to find best depth limit $l$.
  - Key idea: use **Depth-limited search** as subroutine, with increasing $l$.

  For $l = 0$ to $\infty$
  - depth-limited-search to level $l$
  - if it succeeds
    - then return solution

  - **Complete & optimal:** Goal is always found at depth $d$, the depth of the shallowest goal-node.

  Could this possibly be efficient?

Nodes constructed at each deepening

- Depth 0: 0 (Given the node, doesn’t construct it.)
- Depth 1: $b^1$ nodes
- Depth 2: $b$ nodes + $b^2$ nodes
- Depth 3: $b$ nodes + $b^2$ nodes + $b^3$ nodes
- ...

Total nodes constructed:

- Depth 0: 0 (Given the node, doesn’t construct it.)
- Depth 1: $b^1$ = $b$ nodes
- Depth 2: $b$ nodes + $b^2$ nodes
- Depth 3: $b$ nodes + $b^2$ nodes + $b^3$ nodes
- ...

Suppose the first solution is the last node at depth 3:

- Total nodes constructed:
  - $3b$ nodes + $2b^2$ nodes + $1b^3$ nodes
ID search, Evaluation II: Time Complexity

- More generally, the time complexity is
  \( (d)b + (d-1)b^2 + \ldots + (1)b^d = O(b^d) \)

- As efficient in terms of \( O(\ldots) \) as Breadth First Search:
  \( b + b^2 + \ldots + b^d = O(b^d) \)

ID search, Evaluation III

- Complete: YES (no infinite paths) 😊
- Time complexity: \( O(b^d) \)
- Space complexity: \( O(bd) \)
- Optimal: YES if step cost is 1. 😊

Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Time</td>
<td>( b^d )</td>
<td>( b^m )</td>
<td>( b^l )</td>
</tr>
<tr>
<td>Space</td>
<td>( b^d )</td>
<td>( bm )</td>
<td>( bl )</td>
</tr>
<tr>
<td>Optimal?</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>