Uninformed Search Strategies

AIMA 3.3-3.4

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

Art: Formulating a Search Problem

Decide:

- Which properties matter & how to represent
  - Initial State, Goal State, Possible Intermediate States
- Which actions are possible & how to represent
  - Operator Set: Actions and Transition Model
- Which action is next
  - Path Cost Function

Formulation greatly affects combinatorics of search space and therefore speed of search

Hard subtask: Selecting a state space

- Real world is absurdly complex
  - State space must be abstracted for problem solving
- (abstract) State = set (equivalence class) of real world states
- (abstract) Action = equivalence class of combinations of real world actions
  - e.g. Arad → Zerind represents a complex set of possible routes, detours, rest stops, etc
  - The abstraction is valid if the path between two states is reflected in the real world
- Each abstract action should be “easier” than the real problem

Example: Missionaries & Cannibals

Three missionaries and three cannibals come to a river. A rowboat that seats two is available. If the cannibals ever outnumber the missionaries on either bank of the river, the missionaries will be eaten. (AIMA problem 3.9)

How shall they cross the river?

Formulation: Missionaries & Cannibals

- How to formalize:
  - Initial state: all M, all C, and boat on one bank
  - Actions: ??
  - Transition Model??
  - Goal test: True if all M, all C, and boat on other bank
  - Cost: ??

Remember:

- Representation:
  - States: Which properties matter & how to represent
  - Actions & Transition Model: Which actions are possible & how to represent
  - Path Cost: Deciding which action is next
Missionaries and Cannibals

States: (CL, ML, BL)
Initial: 331  Goal: 000

Actions:

<table>
<thead>
<tr>
<th>Travel Across</th>
<th>Travel Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>-101</td>
<td>101</td>
</tr>
<tr>
<td>-201</td>
<td>201</td>
</tr>
<tr>
<td>-011</td>
<td>011</td>
</tr>
<tr>
<td>-021</td>
<td>021</td>
</tr>
<tr>
<td>-111</td>
<td>111</td>
</tr>
</tbody>
</table>

Outline for today’s lecture

- Formulating Search Problems – An Example
- 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 $s_j$ to another state $s_k$
- **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)
  - 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
  - Tree search treats different paths to the same node as distinct

Review: Generalized tree search

- A state is a (representation of a) physical configuration
- A node is a data structure constituting part of a search tree
- Also includes parent, children, depth, path cost $g(x)$
- Here node= $<state$, parent-node, children, action, path-cost, depth$>
- States do not have parents, children, depth or path cost!

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
  - Also includes parent, children, depth, path cost $g(x)$
- 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
    - creates new nodes, fills in the various fields
8-Puzzle Search Tree

- (Nodes show state, parent, children - leaving Action, Cost, Depth Implicit)
- Suppressing useless “backwards” moves

Problem: Repeated states

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

Solution: Graph Search!

- Graph search
  - Optimal but memory inefficient
  - Simple Mod from 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

- Formulating Search Problems – An Example
- 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 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 \(\infty\))

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
      - e.g. 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.INITIAL-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)

From Figure 3.11 Breadth-first search (ignores loops, repeated nodes)

**Properties of breadth-first search**

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

- **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 milliseconds</td>
<td>107 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
<td>11 milliseconds</td>
<td>16.6 megabytes</td>
</tr>
<tr>
<td>6</td>
<td>10^6</td>
<td>1.1 seconds</td>
<td>1 gigabyte</td>
</tr>
<tr>
<td>8</td>
<td>10^8</td>
<td>2 minutes</td>
<td>103 gigabytes</td>
</tr>
<tr>
<td>10</td>
<td>10^10</td>
<td>3 hours</td>
<td>10 terabytes</td>
</tr>
<tr>
<td>12</td>
<td>10^12</td>
<td>3.5 years</td>
<td>1 petabyte</td>
</tr>
<tr>
<td>14</td>
<td>10^14</td>
<td>99 petabytes</td>
<td></td>
</tr>
</tbody>
</table>

Fig 3.13 Assumes b=10, 1M nodes/sec, 1000 bytes/node
Review: Depth-first search

- **Idea:**
  - Expand *deepest* unexpanded node

- **Implementation:**
  - *frontier* is LIFO (Last-In-First-Out) Queue:
    - Put successors at the front of *frontier* successor list.

Properties of depth-first search

- **Complete?** No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
  - **Complete in finite spaces**

- **Time?** $O(b^m)$: terrible if $m$ is much larger than $d$
  - but if solutions are dense, may be much faster than breadth-first

- **Space?** $O(b^m)$, i.e., linear space!

- **Optimal?** No

  $b$: maximum branching factor of search tree
  $d$: depth of the least cost solution
  $m$: maximum depth of the state space (= $\infty$)

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 3.4.4-3.4.5)**
  - 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 $\ell$*
  - i.e. nodes at depth $\ell$ *have no successors.*
  - No infinite-path problem!

  - **If $\ell = d$ (by luck!), then optimal**
    - But:
      - *If $\ell < d$ then incomplete* ☹
      - *If $\ell > d$ then not optimal* ☹

  - **Time complexity:** $O(b^\ell)$
  - **Space complexity:** $O(b\ell)$ ☝
Iterative deepening search

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

\[
\text{For } l = 0 \text{ to } \infty \text{ do }
\text{depth-limited-search to level } l
\text{ if it succeeds then return solution}
\]
- 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:
\[
3*b \text{ nodes} + 2*b^2 \text{ nodes} + 1*b^3 \text{ nodes}
\]

ID search, Evaluation II: Time Complexity

- More generally, the time complexity is

\[
O(d)b + (d-1)b^2 + ... + (1)b^d = O(b^d)
\]
- As efficient in terms of \( O(\ldots) \) as Breadth First Search:

\[
b + b^2 + ... + b^d = O(b^d)
\]

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-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>
<td>YES</td>
</tr>
<tr>
<td>Time</td>
<td>( b^d )</td>
<td>( b^d )</td>
<td>( b^l )</td>
<td>( b^d )</td>
</tr>
<tr>
<td>Space</td>
<td>( b^d )</td>
<td>( b^m )</td>
<td>( b^l )</td>
<td>( bd )</td>
</tr>
<tr>
<td>Optimal?</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>