Search II: Uninformed Search Strategies

AIMA 3.4

Iterative-Deepening Search
Bidirectional Search
Uniform-Cost Search

The Goat, Cabbage, Wolf Problem

(From xkcd.com)

Review: Search Strategies

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

- Time/space complexity measurements
  - $b$, maximum branching factor of search tree
  - $d$, depth of the shallowest goal node
  - $m$, maximum length of any path in the state space (potentially $\infty$)

Search Conundrum

- Breadth-first
  - Complete
  - uses $O(b^d)$ space
- Depth-first
  - Not complete unless $m$ is bounded
  - Uses $O(b^m)$ time; terrible if $m \gg d$
  - 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.
  - Solves the 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: $O(b\ell)$
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
- ...

Iterative deepening search

- A general strategy to find best depth limit \(\ell\)
  - Key idea: use Depth-limited search as subroutine, with increasing \(\ell\).
  - Complete & optimal: Goal is always found at depth \(d\), the depth of the shallowest goal-node.

- Combines benefits of DF-search & BF-search

Iterative Deepening Search

For \(d = 0\) to \(\infty\) do
  - depth-limited-search to level \(d\)
    - if it succeeds
      - return solution

(for details see full algorithm in AIMA pp 88-90)

Could this possibly be efficient?

Total nodes constructed:

- 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
- ...

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

Total nodes constructed (assuming \(b=10\)):

\[
3 \times b + 2 \times b^2 + b^3 = 30 + 200 + 1000 = 1230
\]

ID search, Evaluation II: Time Complexity

- More generally, for a search tree with shallowest solution at depth \(d\), worst case, the time complexity is
  \[
  N(\text{IDS}) = (d+1) + (d)b + (d-1)b^2 + \ldots + (1)b^d = O(b^d)
  \]

- As efficient as Breadth First Search:
  \[
  N(\text{BFS}) = b + b^2 + b \ldots + b^d = O(b^d)
  \]

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-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>$b^m$</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>$b^d$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>YES*</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

Very briefly: Bidirectional search

- Two simultaneous searches from start and goal.
  - Motivation: $b^{d/2} + b^{d/2} < b^d$
  - Check whether the node belongs to the other frontier before expansion.
  - Space complexity is the most significant weakness.
  - Complete and optimal if both searches are Breadth-First.

How to search backwards?

- The predecessor of each node must be efficiently computable.
  - Works well when actions are easily reversible.

“Uniform Cost” Search

Motivation: Romanian Map Problem

- All our search methods so far assume step-cost = 1
  - This isn’t always true

$g(N)$: the path cost function

- If all moves equal in cost:
  - $g(N) = \text{depth}(N)$
  - Equivalent to what we’ve been assuming so far

- Assigning a (potentially) unique cost to each step
  - $N_0, N_1, N_2, N_j$ = nodes visited on path $p$
  - $C(i,j)$: Cost of going from $N_i$ to $N_j$
  - $g(N_j) = C(0,1) + C(1,2) + C(2,3)$
Uniform-cost search (UCS)

- Extension of BF-search:
  - Expand node with lowest path cost
- Implementation: frontier = queue ordered by g(n)
- Differs from BF-search:
  - Tests if a node is a goal state when it is selected for expansion, not when it is added to the frontier.
  - Updates a node on the frontier if a better path to the same state is found.
  - So always enqueues nodes before checking whether they are goals.
  - WHY???
- (Dijkstra’s algorithm just UCS without goal)

Summary of algorithms (for notes)

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-cost</th>
<th>Depth-Limited</th>
<th>Iterative deepening</th>
<th>Bidirectional search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Time</td>
<td>( b^n )</td>
<td>( b^{1+C/e} )</td>
<td>( b^m )</td>
<td>( b^d )</td>
<td>( b^{d/2} )</td>
</tr>
<tr>
<td>Space</td>
<td>( b^d )</td>
<td>( b^{1+C/e} )</td>
<td>( b^m )</td>
<td>( b^d )</td>
<td>( b^{d/2} )</td>
</tr>
<tr>
<td>Optimal?</td>
<td>YES*</td>
<td>YES*</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

- If all costs in Uniform-cost search are 1, then time and space both \( O(b^{d+e}) \)