Uninformed Search (Ch. 3-3.4)

Come on, I need answers...
Announcements

HW due tonight

Writing 1 due next week
Search algorithm

To search, we will build a tree with the root as the initial state

```
function tree-search(root-node)
    fringe ← successors(root-node)
    while ( notempty(fringe) )
        {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         fringe ← insert-all(successors(node),fringe) }
    return failure
end tree-search
```

Any problems with this?
Search algorithm

We can remove visiting states multiple times by doing this:

```plaintext
def tree-search(root-node):
    fringe ← successors(root-node)
    explored ← empty
    while (not empty(fringe))
        {node ← remove-first(fringe)
            state ← state(node)
            if goal-test(state) return solution(node)
            explored ← insert(node, explored)
            fringe ← insert-all(successors(node), fringe, if node not in explored)
        }
    return failure
end tree-search
```

But this is still not necessarily all that great...
The search algorithms metrics/criteria:
1. Completeness (does it terminate with a valid solution)
2. Optimality (is the answer the best solution)
3. Time (in big-O notation)
4. Space (big-O)

\[ b = \text{maximum branching factor} \]
\[ d = \text{minimum depth of a goal} \]
\[ m = \text{maximum length of any path (depth of tree)} \]
Breadth first search checks all states which are reached with the fewest actions first (i.e. will check all states that can be reached by a single action from the start, next all states that can be reached by two actions, then three...).
Breadth first search

(see: https://www.youtube.com/watch?v=5UfMU9TsoEM)
(see: https://www.youtube.com/watch?v=nI0dT288VLs)
Breadth first search

BFS can be implemented by using a simple FIFO (first in, first out) queue to track the fringe/frontier/unexplored nodes

Metrics for BFS:
Complete (i.e. guaranteed to find solution if exists)
Non-optimal (unless uniform path cost)
Time complexity = $O(b^d)$
Space complexity = $O(b^d)$
Breadth first search

Exponential problems are not very fun, as seen in this picture:

<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>.11 milliseconds</td>
<td>107 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>11,110</td>
<td>11 milliseconds</td>
<td>10.6 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>13 days</td>
<td>1 petabyte</td>
</tr>
<tr>
<td>14</td>
<td>$10^{14}$</td>
<td>3.5 years</td>
<td>99 petabytes</td>
</tr>
<tr>
<td>16</td>
<td>$10^{16}$</td>
<td>350 years</td>
<td>10 exabytes</td>
</tr>
</tbody>
</table>

Figure 3.13  Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.
Uniform-cost search also does a queue, but uses a priority queue based on the cost (the lowest cost node is chosen to be explored)
Uniform-cost search

The only modification is when exploring a node we cannot disregard it if it has already been explored by another node.

We might have found a shorter path and thus need to update the cost on that node.

We also do not terminate when we find a goal, but instead when the goal has the lowest cost in the queue.
Uniform-cost search

UCS is..

1. Complete (if costs strictly greater than 0)
2. Optimal

However....

3&4. Time complexity = space complexity = $O(b^{1+C*/\min\text{path cost}})$, where $C^*$ cost of optimal solution (much worse than BFS)
Depth first search

DFS is same as BFS except with a FILO (or LIFO) instead of a FIFO queue
Depth first search

Metrics:
1. Might not terminate (not correct) (e.g. in vacuum world, if first expand is action L)
2. Non-optimal (just... no)
3. Time complexity = $O(b^m)$
4. Space complexity = $O(b*m)$

Only way this is better than BFS is the space complexity...
Depth limited search

DFS by itself is not great, but it has two (very) useful modifications

Depth limited search runs normal DFS, but if it is at a specified depth limit, you cannot have children (i.e. take another action)

Typically with a little more knowledge, you can create a reasonable limit and makes the algorithm correct
Depth limited search

However, if you pick the depth limit before \( d \), you will not find a solution (not correct, but will terminate)
Iterative deepening DFS

Probably the most useful uninformed search is **iterative deepening DFS**

This search performs depth limited search with maximum depth 1, then maximum depth 2, then 3... until it finds a solution.
Iterative deepening DFS
Iterative deepening DFS

The first few states do get re-checked multiple times in IDS, however it is not too many

When you find the solution at depth $d$, depth 1 is expanded $d$ times (at most $b$ of them)

The second depth are expanded $d-1$ times (at most $b^2$ of them)

Thus $d \cdot b + (d - 1) \cdot b^2 + \ldots + 1 \cdot b^d = O(b^d)$
Iterative deepening DFS

Metrics:
1. Complete
2. Non-optimal (unless uniform cost)
3. $O(b^d)$
4. $O(b^d)$

Thus IDS is better in every way than BFS (asymptotically)

Best theoretical uninformed we will talk about
Bidirectional search

Bidirectional search starts from both the goal and start (using BFS) until the trees meet

This is better as $2 \times (b^{d/2}) < b^d$

(the space is much worse than IDS, so only applicable to small problems)
## Summary of algorithms

**Fig. 3.21, p. 91**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening DLS</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes[a]</td>
<td>Yes[a,b]</td>
<td>No</td>
<td>No</td>
<td>Yes[a]</td>
<td>Yes[a,d]</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\epsilon})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\epsilon})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes[c]</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes[c]</td>
<td>Yes[c,d]</td>
</tr>
</tbody>
</table>

There are a number of footnotes, caveats, and assumptions. See Fig. 3.21, p. 91.

[a] complete if $b$ is finite

[b] complete if step costs $\geq \epsilon > 0$

[c] optimal if step costs are all identical
(Also if path cost non-decreasing function of depth only)

[d] if both directions use breadth-first search
(Also if both directions use uniform-cost search with step costs $\geq \epsilon > 0$)