Evaluating Uninformed Search

1 Summary

By the end of this lecture, you should be able to fill in the entries in this table:

Strategy	Complete?	Optimal?	Time complexity	Space complexity
Breadth-first				
Depth-first				
Depth-bounded				
Iterative-deepening				
Least-cost				

Notation

b	the branching factor of the state space (see below)
d	the length of the shortest solution path
m	the maximum depth of the search tree (assuming it's finite)
l	the depth-bound used in depth-bounded search
$g(n)$	the cost of the path from the start node to node n

To make our mathematical analysis easier, we assume that we are dealing with a search tree in which every node has the same number of successors, b. This, of course, is an idealisation. b is called the *branching factor* of the search tree (or state space). We have been assuming, and we continue to assume, that b is finite.

2 Breadth-First Search

Complete? Breadth-first search is complete. *Why? (By the way, what happens when there is no solution path?)*

Optimal? Breadth-first search is optimal (in the sense that the first solution path that it finds (if any) will be the shortest solution path). *Why?*

Time complexity At level 0, we generate one node, the start node. At level 1, there are b successors of the start node. Each of these b successors has b successors, so at level 2 there are b^2 nodes. Each of the nodes at level 2, has b successors, so there are b^3 nodes at level 3. If the first solution path is of length d, then the maximum number of nodes we would generate is

$$1 + b + b^2 + b^3 + \ldots + b^{d-1} + b^d$$

The *worst* case time complexity is therefore $O(b^d)$.

In the *best* case, we wouldn't need to expand the whole of level d; we would need to expand only one node at this level, i.e. the first one we try gives us a solution, so the best case is:

$$1 + b + b^2 + b^3 + \ldots + b^{d-1} + 1$$

Assuming that each node at depth d is equally likely to be a goal node and there is only one goal node, then the *average* time complexity is

$$1 + b + b^2 + b^3 + \ldots + b^{d-1} + \frac{1 + b^d}{2}$$

Space complexity The space complexity is the same as the time complexity. We have to maintain all of the tree that we have so far generated, so that each path can be extended level by level.

Exponential *time* complexity is very bad news. But exponential *space* complexity is a disaster. This search strategy can only be used on trivial state spaces.

3 Depth-First Search

Complete? Depth-first search is not complete. *Why?*

Optimal? Depth-first search is not optimal. *Why?*

Time complexity To give complexity results, we'll assume the search tree is finite and has maximum depth m. In the worst case (where $d = m$ and, what's more, the goal node is the last node on level m), depth-first search will visit all nodes in the tree:

$$1 + b + b^2 + b^3 + \ldots + b^m$$

i.e. $O(b^m)$. Note how this is, in general, worse than breadth-first: because the paths that this strategy explores before it finds this goal node must be explored not just to level d but to their full extent, m.

The best-case time complexity is just $d + 1$, i.e. $O(d)$: if the first path we explore is the one that contains the shallowest goal, then we walk straight down this path to the goal, with no time spent on fruitless paths.

The average-case is harder to determine so I'll just state it rather than try to explain it: again if we assume each node is just as likely to be the goal, it turns out to be roughly the same as that for breadth-first search.

All this sounds bad, but, in practice, in many state spaces, depth-first search will find a solution path more quickly than breadth-first search. *In what kinds of state spaces might it find a solution path more quickly?*

Space complexity Depth-first search has excellent space complexity. It needs to store the start node. Then at level 1, it stores b successors of the start node. Then at level 2, it stores b successors of *one* of these nodes. At level three, the b successors of *one* of the level 2 nodes are stored. And this goes on, in the worst case, until the path can be extended no further (level m):

$$1 + \underbrace{b + b + b + \ldots + b}_{m \text{ times}}$$

When it moves on to another path, the space that was being used for the path that it can extend no further can be reclaimed and reused. Hence, the worst-case space complexity is $O(bm)$. Its space requirement is linear.

4 Depth-Bounded Search

Complete? *When is depth-bounded search complete?*

Optimal? Depth-bounded search is not optimal for much the same reasons as depth-first search is not optimal.

Time complexity The analysis here is the same as that for depth-first search, except that the longest paths the algorithm will explore are now of length l (the depth-bound). Hence, the time complexity is $O(b^l)$.

Space complexity Similarly, the space complexity is $O(bl)$.

5 Iterative-Deepening

Iterative-deepening is probably the best all-round uninformed search strategy for state spaces that have uniform cost actions. It manages to combine the strengths of breadth-first and (depth-bounded) depth-first search.

Complete? Iterative-deepening search is complete. *Why?*

Optimal? Iterative-deepening search is optimal (again in the sense that the first solution path that it finds (if any) will be the shortest solution path). *Why?*

Time complexity People worry about the wasteful repeated computation carried out by iterative-deepening search. Let's analyse it.

If the shallowest solution is at depth d, then iterative-deepening does $d+1$ searches, each having the complexity we saw for depth-bounded search:

$$1 + (1 + b) + (1 + b + b^2) + (1 + b + b^2 + b^3) + \ldots + (1 + b + b^2 + b^3 + \ldots + b^d)$$

Re-arranging gives:

$$(d + 1) + db + (d - 1)b^2 + (d - 2)b^3 + \ldots + 3b^{d-2} + 2b^{d-1} + b^d$$

So, the time complexity is the same as breadth-first search: in the worst case, it is $O(b^d)$.

Some people would continue to worry about the *practical* cost of the repeated effort. The thing to remember here is that the levels of the tree that get re-generated most often are the ones with *fewest* nodes in them. At level d, there are many nodes but they're visited only once; at level $d - 1$, there are far fewer nodes, and they're visited only twice; at level 2 there are only b^2 nodes and they're visited $(d - 1)$ times; at level 1, there are b nodes and they're visited d times; the start node is visited $d + 1$ times.

Space complexity Iterative-deepening search is a repeated form of (depth-bounded) depth-first search. Therefore, its space complexity is like that of depth-bounded and depth-first search, in this case $O(bd)$.

6 Least-Cost Search

Complete? Least-cost search can be complete. But there's a requirement on the path cost function g. *What's the requirement?*

Optimal? The same requirement is needed to guarantee optimality. *Explain why, with this requirement satisfied, least-cost search is optimal.*

Time complexity Least-cost search may have to explore the whole state space. Hence, its worst-case time complexity is $O(b^m)$. Of course, you hope that, in practice, guiding the search using the path costs will result in a more focused search than the plodding breadth-first search.

Space complexity Similarly, its space complexity is $O(b^m)$.