1 Notes

Parts of a search problem: start state, transition model, goal test, path cost

What search algorithms have in common:
There’s always a search tree involved.
1. Root node is annotated with start state.
2. Successively expand nodes.
3. When you expand a goal node, you’re done.

Dimensions of variation

1. Whether you remember states you’ve visited before
2. How you decide which node to expand
   - Uninformed: breadth-first, depth-first, uniform cost, iterative deepening
   - Informed:
     - Best-first (minimize some metric $f(n)$)
     - A*: $f(n) = g(n) + h(n)$
     - Greedy: $f(n) = h(n)$
     - Beam search (like greedy, but prune paths)

2 Questions

1. Do search algorithms typically represent the search tree explicitly?
2. Can two nodes of a search tree be annotated with the same state?
3. Consider the following undirected graph: \{(A,B) (B,C) (C,D) (D,E) (E,B) (D,F)\}. Suppose you’re trying to find a path from A to F. If you don’t mark graph nodes you’ve visited, is a BFS guaranteed to terminate? DFS? Iterative deepening? What if you do mark nodes?
4. With the same graph, suppose you’re trying to find a path from A to G, but G does not actually exist. Using a BFS with no marking of visited states, how many nodes of the search tree will you expand? (Don’t confuse nodes in the search tree and nodes in the graph!)
5. In the worst case, how much space does it take to run a BFS on a graph with $n$ nodes? What about DFS? What if your DFS implementation is tail-recursive?
6. Find a graph that would take more memory to search using DFS than BFS.
7. What data structures would you use to implement (a) BFS (b) DFS (c) uniform cost search (d) best-first search?

8. Dijkstra’s algorithm is a specialization of one of the above search algorithms. Which one, and what is the specialization?

9. Justin T. writes an implementation of A*. His heuristic is admissible and consistent. He always knows which node to expand because he searches the open list and picks the node whose evaluation function is minimal. When his algorithm expands a node of the search tree, it checks to see whether any of the children of that node matches the goal node. If so, it reports the path it’s found to the goal. If not, all the children go on the open list. Justin’s professor fails him! Why?

10. Paris H. suggests the following proof of the optimality of A*. Suppose A* terminates by expanding node \( w \) of the search tree which satisfies the goal test, resulting in a path \( P \) which gets reported as the best. The cost of \( P \) equals \( f(w) \), the evaluation function applied to \( w \), so it must be less than or equal to \( f(n) \) for all nodes \( n \) we’ve previously seen. Thus for any previous node \( n \) of the the search tree, the cost of \( P \) is less than the best path starting with the route to \( n \). Letting \( n \) equal the start node, \( P \)’s cost is less than or equal to the best possible cost from the start node. Obviously it’s not less, so it must be equal. Thus \( P \) is the best path from the start node to a goal node. Paris’s professor fails her! Why?

11. Jeff S., former CEO, in one last desperate attempt to impress the professor, tries to prove that consistency implies that if A* expands a node \( q \) corresponding to state \( S \), and then later expands another node \( q_2 \) corresponding to the same state \( S \), the path \( q_2 \) represents was more costly. He reasons as follows, where \( g \) is path cost, \( h \) is the heuristic, and \( f \) is the evaluation function:

\[
g(q) + h(q) = f(q) \leq f(q_2) = g(q_2) + h(q_2)
\]  

(1)

Since \( h(q) = h(q_2) \), \( g(q) \leq g(q_2) \). Unbelievably, the professor fails Jeff too! Why?

12. In order for A* to be complete, must the heuristic be admissible? Consistent? (Be careful.)

13. In order for A* to be optimal, must the heuristic be admissible? Consistent? (Be careful.)


15. Prove that the first path Dijkstra’s algorithm finds to a given node is the shortest path to that node.

16. Under what circumstance is the same thing true of A*? Prove it.

17. Why is consistency desirable?

18. What would be a good heuristic for solving Sudoku by search?

19. How would you implement Dijkstra’s algorithm in \( O(\epsilon \log v) \) time, where \( v \) is the number of vertices in the graph, and \( \epsilon \) is the number of edges?

20. Find a graph of \( n \) nodes where a search with BFS or DFS is \( O(n) \) but a search with iterative deepening is \( O(n^2) \).