State Space Search Self-Check

605.645 – Artificial Intelligence

The purpose of this self-check is to make sure you understand key concepts for the algorithms presented during the module and to prepare you for the programming assignment. As you work through problems, you should always be thinking “how would I do this in code? What basic data structures would I need? What operations on those basic data structures?”

ALL self checks are to be done by hand. No coding.

State Space Search

State Space Search plays a fundamental role in Symbolic AI, which attempts to model human reasoning. State Space Search is typically defined in terms of States, Actions, Transitions and (possibly) Costs. Additionally, for *informed* search you will need a *heuristic* function.

The challenge when solving a real world problem using State Space Search is to identify all of these components, some of which will be *implicit*. Additionally, your program may need to store *metadata* (bookkeeping) about states as the algorithm progresses.

There are generally two types of State Space Search problems:

1. **We care about the path. We may care about the final state.**

   We know what the goal state (or states) looks like but we don’t know how to get there. A good example of this kind of problem is the Farmer, Wolf, Goat, & Cabbage problem. The goal looks like “everyone in one piece on the West side of the river” but we don’t know how to achieve the goal. Thus, the path through the state space is important.
2. **We care only about the final state.**

   We don’t know what the goal state (or states) looks like but we can tell if we have reached one. We don’t care how we get there. We just want to know what the final result should look like. N-Queens is like this. Who cares what order you place the queens on the board, we just want a final result that has no queen attacking any other queen.

So the big question is, do you care how you got to the solution? If so, you have Case 1 and if you don’t, you have Case 2.

Consider the general pseudocode for State Space Search (using *graph* search):

```
place initial state on frontier
initialize explored list
while frontier is not empty
    current-state := next state on frontier
    return path( current-state) if is-terminal( current-state)
    children := successors( current-state)
    for each child in children
        add child to frontier if not on explored or frontier
    add current-state to explored
return nil
```

Practice

Imagine we have the following raw data for a “microworld” that a robot must navigate:

```
  0 1 2 3
0 S o o o
1 o x x o
2 o o o o
3 o o x G
```

The coordinate system is (x, y) so that (2, 1) is a “x” where “S” is a starting state, “G” is a goal state, “o” is an open position and “x” is a closed/blocked position. The robot’s movement is limited north (up), east (right), south (down), and west (left) so no diagonal moves.

1. What is a *state* in this problem? How would you characterize it (what data is needed)? What is the set of states, S?
2. How is the set of *transitions*, T, related to the successors function? Do we need specify the set of transitions explicitly? If the robot can only move up, down, left or right by one square, what would the successors function produce for the starting state, S?
3. Draw the graph that is generated by your successor function/transitions set. Based on the algorithm above, is the actual graph generated all at once or piecemeal?
4. Solve the above State Space Search problem of moving from S to G using the Depth First Search (DFS) algorithm. Show all of your work including the state of the frontier and explored list. In order for everyone to generate the same search path, always generate children in this order: W, S, E, N (skipping actions that don’t apply).
5. Is this the optimal (lowest cost) path? What is the optimal path?
6. Notice the fold at (0, 2) -> (0, 3) -> (1,3) -> (1, 2) -> (2, 2) instead of going directly from (0, 2) to (1, 2). What caused this? Was it DFS or the order that we placed successor states on the frontier? Is there a different order we can use to place successors on the frontier to fix this problem? Will it work for all placements of G?
7. If we *are* in Case 1, we did nothing that enabled us to recover the path (no bookkeeping). What additional information should we have saved to recover the path?

You should show all your work. For each step, show the frontier (also called the *fringe*) and the explored list. Enumerate the states of the path in order and show the path on the world using “*”.

As you complete this assignment, think about the data structures, algorithms and metadata you would need to implement this as a Python program.

Additional Questions

These questions are not required but it will be worth your while to work through them and/or discuss them in your Discussion Groups.

1. What changes would need to be made to make this Breadth First Search? What path would it find?
2. What changes would need to be made to make this Greedy Search? What path would it find? (Assume an action cost of 1).
3. What changes would need to be made to make this A* Search? What path would it find? (Assume an action cost of 1).