CSCI-B 551: Elemantary Artificial Intelligence

Assignment 0: N-queens and Python

Khandokar Md. Nayem
Student ID: 2000060966
Fall 2016

September 6, 2016

1. Spend some time familiarizing yourself with the code. Write down the precise abstraction that the program is using. In other words, what is the set of valid states, the successor function, the cost function, the goal state definition, and the initial state?

Set of Valid States: Any combination of 0(no rook) and 1(rook) in $(N \times N)$ size board. In each board position, there is only 2 possible combination can happen and they are 0(no rook) and 1(rook). So in $N \times N(N^2)$ size board, the possible valid states are 2^{N^2}, where N = one side of board.

Successor function: Returns the board with a rook added in a specific square or adding no new rook in the old board. From any valid states, successor function generates (N^2) valid states. The given successor function does not check whether there's already a rook in a board block or not.

Cost function: This is an uniform Cost function. Placing one more rook, needs an equal weight. But in a self cycle like- doing nothing also cost the equal weight.

Goal state definition: In the goal state, three conditions have to satisfy. They are,

- There are N rooks in the board in total.
- There are at most 1 rook in each row.
- There are at most 1 rook in each column.

Initial state: The empty board with no rook(all 0) is the initial state of N-rook code.

2. Is the algorithm using BFS or DFS? Switch to the opposite, and explain exactly how to modify the code to do this. What happens now for N=4 and N=8?

The algorithm is using DFS algorithm. Because, in the fringe the successor states are appended at the end. And new states are popped from the end also, just like a stack.

For implementing BFS algorithm, we have to edit inside the nested loop of the `solve()` method. To switch between BFS and DFS algorithm we have to use one of the following,

- `fringe.append(s)` # stack(for DFS)
- `fringe.insert(0,s)` # queue(for BFS)

Here, s is a new valid state of Board generated by successor function.

For N=4, BFS can give solution in around 0.07s. Whereas DFS can't give any solution. Because, DFS algorithm gets caught in a cycle.

For N=8, BFS can't give solution in any decent time. Because BFS checks all of its successor states in fringe which is very large (2^{N^2}). As N increases , branching factor increases too and generate more successor states in each level. However, the cycle problem of DFS is persistent in larger N too. So DFS algorithm goes around like forever.

3. The successor function in the code is defined in a very simplistic way, including generating states that have N+1 rooks on them, and allowing moves that involve not adding a rook at all. Create a new successors() function called successors2() that fixes these two problems. Now does the choice of BFS or DFS matter? Why or why not?

The choice of BFS or DFS matters in this problem. By definition of this N-rooks problem, the Goal state must have N-rooks on the board. In Tree like data structure, this is like a tree of level N. Since DFS very quickly explores the states of N depth, it is most likely to find the solution there sooner than BFS.

On the other hand, BFS checks all the states up to N depth even though there will be no goal states in 0 to $N-1$ depth. In N depth, there will be $N(N-1)(N-2)\dots(N-d) = {}^{N}P_{d}$, where d = *Depth of Tree-1*; states in fringe at a time for BFS algorithm. So when N becomes larger, the number of states in each level increases. And it makes BFS slower. So for this specific N-rooks problem, DFS works better than BFS.

4. Even with the modifications so far, N=8 is still very slow. Recall from class that we could define our state space and successor functions in a smarter way, to reduce the search space to be more tractable.Describe your new abstraction, and then modify the code to implement it with a new successors function called successors3(). Feel free to make other code improvements as well. What is the largest value of N your new version can run on within about 1 minute?

Set of Valid States: Some combination of 0(no rook) and 1(rook) in $(N \times N)$ size board in such a way that, there can be at most 1 rook in each row and column.

Successor function: Returns the valid state of the board with a new rook added in a specific square. And adding no rook is not allowed anymore.

Cost function: This has an uniform Cost function. Placing one more rock, needs an equal weight.

Goal state definition: In the goal state, the board state has to be a valid state and there are N rooks in the board in total.

Initial state: The empty board with no rook(all 0) is the initial state of the N-rook code.

The largest value of N in my new version of N-rooks is N=115(in 59.7833111286s).

5. Now, modify your code so that it solves and displays solutions for both the N-rooks and N-queens problem, using the enhancements above. What is the largest value of N that your new version can solve within 1 minute?

Within 1 minute, my version of code can solve highest N=11(2.26248812675s) for both N-rooks and N-queens problem. However, individually for N-rooks, largest N is N=115 and for N-queens largest N is N=1 in 1 minute.