COMS3005A Lab1 – Report

2652330

06 August 2025

Generating the Game boards

The idea is to do a BFS where the start state is the goal state, we then explore all valid moves of the blank, while avoiding moving into previously explored game states. The order of moves is randomized to ensure that the board is uniformly sampled. When a board with a depth of k is reached, it is selected and returned. Pseudo code for this approach follows:

Algorithm 1 Generate board k moves away

```
function REVERSEBOARD(goalState, k, hashRow, hashCol)
frontier ← queue, visited ← set(goal)
chosen ← None, seen ← 0
frontier.enqueue(GameSnapshot(hashRow, hashCol, 0, goalState))
while frontier not empty do
    current ← frontier.dequeue()
    if current.moves == k then
        seen ← seen +1
        if randint(1,seen) == 1 then
            chosen ← current
        end
    end
    for move ∈ shuffle(getMoves(current.row,current.col)) do
        (newRow,newCol) ← current + Move[move]
        nextConfig ← copy(current.board);
        swap(nextConfig, current.row, current.col, newRow, newCol);
        if nextConfig ∉ visited then
            visited.add(next); frontier.enqueue(GameSnapshot(newRow, newCol,current.moves+1,nextConfig))end
        end
    end
    return chosen
    end function
```

Complexity Graphics

please see the bottom of this document for the relevant complexity graphs

Questions

1) The time complexity of a Breath-First-Search is given by $O(b^d)$, where the branching factor b is $b \in [1,4]$ and the depth d is given by k(number of moves to solve that configuration) resulting in the complexity being $O(b^k)$. The space

complexity grows as the number of nodes are expanded which occurs whenever a new configuration is visited therefore the space complexity is also given by $(O(b^k)$

2) There 9 unique cells on the board which can be arranged in 9! different configurations, where not every configuration is solvable, unsolvable configurations would not contribute to complexity of the BFS. Randomly placing digits and the blank also does not allow a good measure of how the algorithm performs the deeper or longer the algorithm has to search in a tree. By having k moves, we can repeat the algorithm on a particular k to get an average of the performance of the algorithm at depth k. Thereby we can gauge how the algorithm may perform as we increase the value of k as k is a controlled parameter.

3) You ensure that you are not moving the board around in cycles, you also ensure that if you have seen a state before you cannot enter that state again. This ensures that the board cannot be solved in fewer than k moves.

4)Time complexity may vary a lot at larger values of k, while the space complexity is fairly similar. Depending on the position of where the blank space is, this can effect the number of branches that come from a specific node, a node can have between 2 and 4 branches. The branch factor can vary even is the value of k is the same. Bottlenecks may occur within the same search tree.

5)The space and time complexity calculated is $O(1.65^k)$, when taking into account that the branching factor is not constant and ranges between 2 and 4 we can see that the theoretical complexity is very accurate. The actual complexity does not have a branching factor of 4 because the tree is being pruned using a visited data structure and an available spaces data structure so that we do not unnecessarily expand nodes wasting computation time.

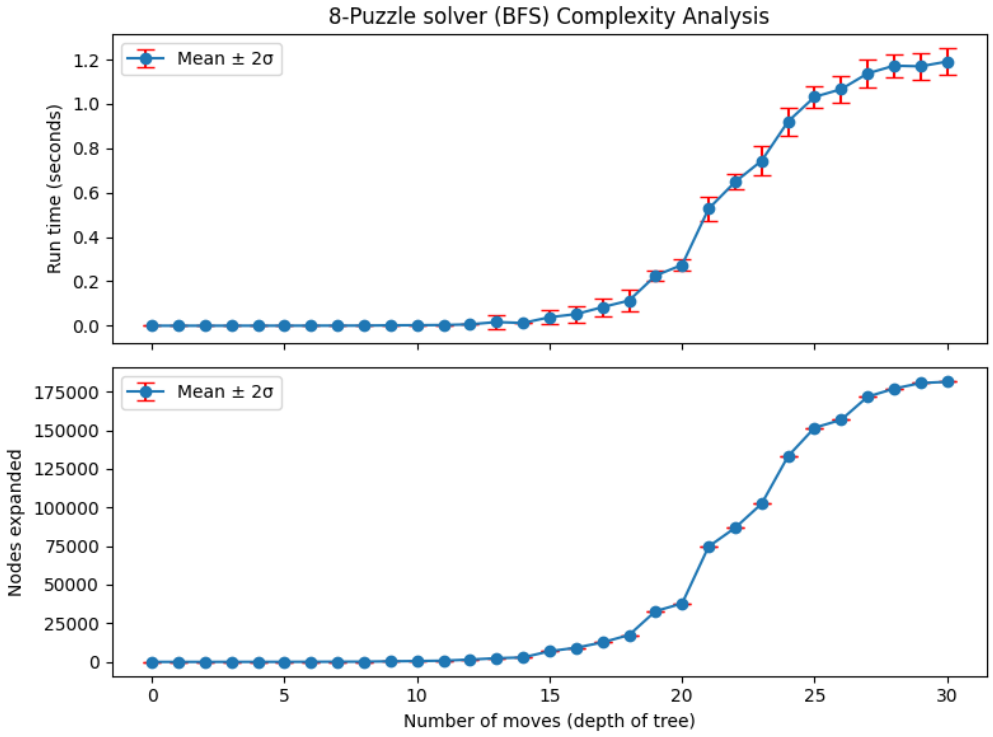

Figure 1: Time and Space Complexity of the BFS showing ±2 deviations