COMP6231 COURSEWORK: PART1

APPROACH

This report aims to demonstrate an implementation of four search algorithms in the blocksworld tile puzzle. Specifically, it will show if it is possible for every algorithm to reach a solution to the problem, how they scale with the problem difficulty, their strength and weaknesses and how they can be tweaked in order to find a faster solution.

The method followed to approach the problem:

1. Implementing the BFS, DFS, IDS and A* algorithms as tree searches.
   The search methods implemented as in the pseudocodes described in the book *"Artificial Intelligence: A modern approach"[1]*.

2. Trying the actions of the agent in a specific sequence and in a randomized order.
   At first, the successors of each node were in a specific order coming from the actions of the agent in the order " ↑ ← ↓ → " (if the action was allowed) and then the order was randomized.

3. Converting the trees to graphs.
   This means that the node would not add as a successor a node whose state is already visited or is about to be visited.

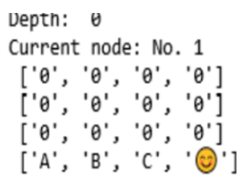
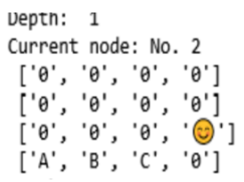
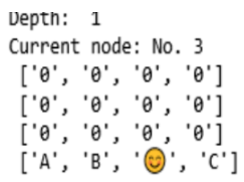
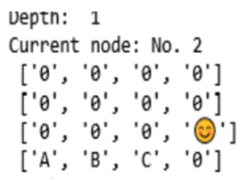
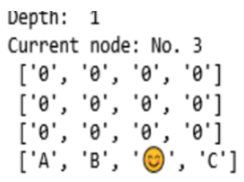
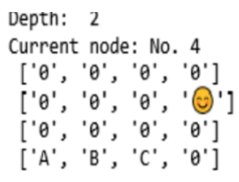
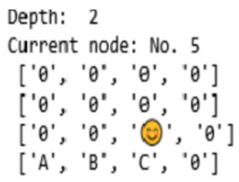
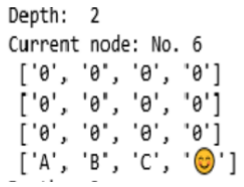
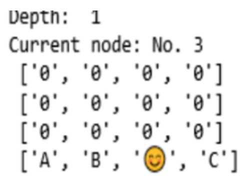
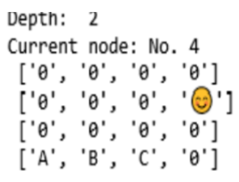
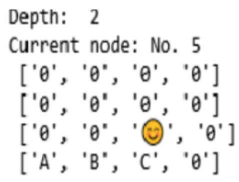
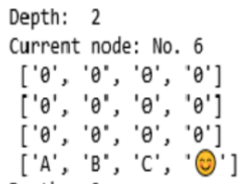
4. As a metric time and space complexity was used.
   Time complexity is equal to the number of nodes generated until the solution (if it is found) is found. Space complexity is the number of nodes in the frontier in the case of a tree and the number of nodes in the frontier and in the visited list in the case of a graph.

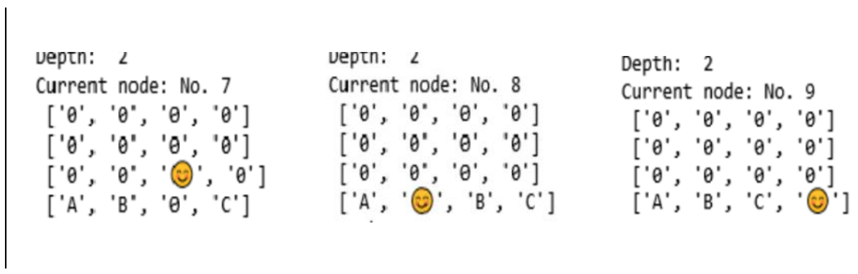
EVIDENCE OF THE 4 SEARCH METHODS IN OPERATION

BREADTH FIRST SEARCH

In breadth first search, a node is expanded first, then all of the successors and so on. Every node is added in the frontier in a FIFO queue, which means older nodes get expanded first.

BFS in this problem is complete because the branching factor in each node is finite so the solution will be found. Also, because the step cost is the same for every node, the solution will be optimal too. The solution will be in the shallowest depth.

Exploration order	Frontier (←FIFO queue←)			
 Depth: 0 Current node: No. 1 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['A', 'B', 'C', '😊']	 Depth: 1 Current node: No. 2 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '😊'] ['A', 'B', 'C', '0']	 Depth: 1 Current node: No. 3 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['A', 'B', '😊', 'C']		
 Depth: 1 Current node: No. 2 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '😊'] ['A', 'B', 'C', '0']	 Depth: 1 Current node: No. 3 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['A', 'B', '😊', 'C']	 Depth: 2 Current node: No. 4 ['0', '0', '0', '0'] ['0', '0', '0', '😊'] ['0', '0', '0', '0'] ['A', 'B', 'C', '0']	 Depth: 2 Current node: No. 5 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '😊', '0'] ['A', 'B', 'C', '0']	
	 Depth: 2 Current node: No. 6 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['A', 'B', 'C', '😊']			
 Depth: 1 Current node: No. 3 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['A', 'B', '😊', 'C']	 Depth: 2 Current node: No. 4 ['0', '0', '0', '0'] ['0', '0', '0', '😊'] ['0', '0', '0', '0'] ['A', 'B', 'C', '0']	 Depth: 2 Current node: No. 5 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '😊', '0'] ['A', 'B', 'C', '0']	 Depth: 2 Current node: No. 6 ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['0', '0', '0', '0'] ['A', 'B', 'C', '😊']	

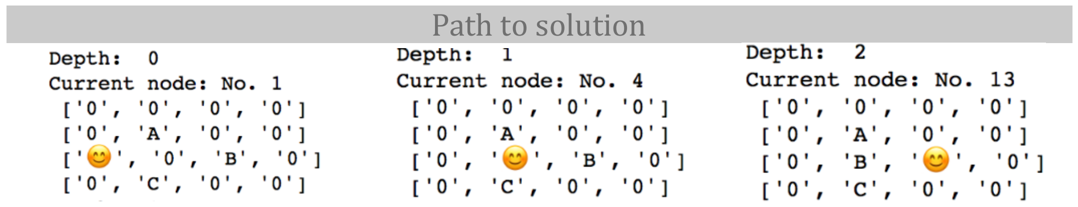


```
Depth:  2
Current node: No. 7
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '😊', '0']
 ['A', 'B', '0', 'C']
```

```
Depth:  2
Current node: No. 8
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', '😊', 'B', 'C']
```

```
Depth:  2
Current node: No. 9
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
```

Figure 1. Left is the current node (the output console from Figure 1). Right are the nodes in the frontier to be explored.



Path to solution

```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['😊', '0', 'B', '0']
 ['0', 'C', '0', '0']
```

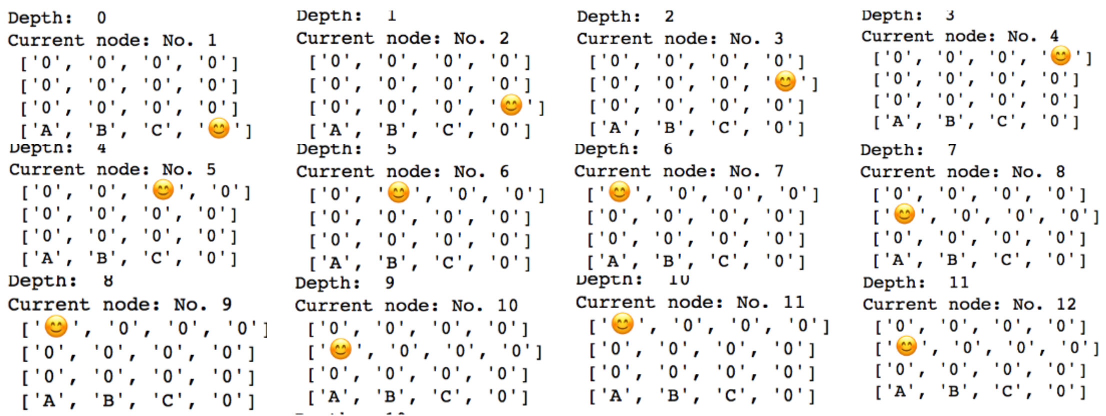
```
Depth:  1
Current node: No. 4
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['0', '😊', 'B', '0']
 ['0', 'C', '0', '0']
```

```
Depth:  2
Current node: No. 13
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['0', 'B', '😊', '0']
 ['0', 'C', '0', '0']
```

Figure 2. Modifying root node in order to find a solution.

DEPTH FIRST SEARCH

Depth first search, in contrast to BFS, expands the deepest node in the tree first, if it is finite, and then explores the other branches. Its frontier works as a LIFO queue. Since, it does not store the visited nodes it is obvious that DFS in our problem will infinite loop as soon as it finds a wall. A weakness of this search is that explores straight to deepest node, that means that for infinite depth the algorithm is not complete nor optimal.



```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
```

```
Depth:  1
Current node: No. 2
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['A', 'B', 'C', '0']
```

```
Depth:  2
Current node: No. 3
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  3
Current node: No. 4
 ['0', '0', '0', '😊']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  4
Current node: No. 5
 ['0', '0', '😊', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  5
Current node: No. 6
 ['0', '😊', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  6
Current node: No. 7
 ['😊', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  7
Current node: No. 8
 ['0', '0', '0', '0']
 ['😊', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  8
Current node: No. 9
 ['😊', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

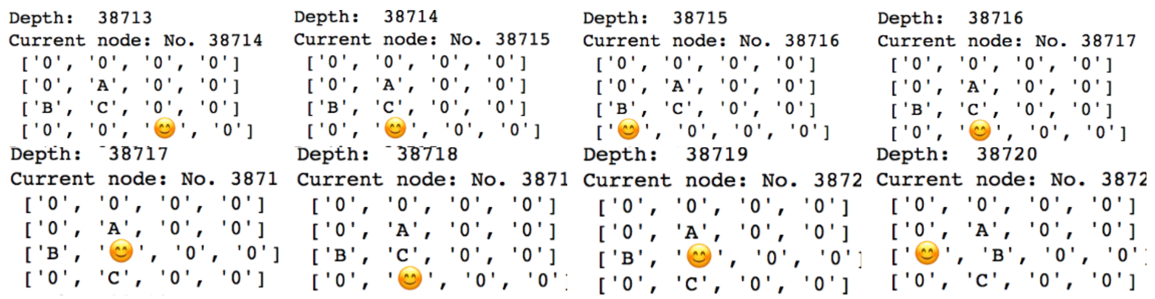
```
Depth:  9
Current node: No. 10
 ['0', '0', '0', '0']
 ['😊', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  10
Current node: No. 11
 ['😊', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  11
Current node: No. 12
 ['0', '0', '0', '0']
 ['😊', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

Figure 3. Stuck in loop in DFS tree following the specific action order "UP, LEFT, DOWN, RIGHT".

A minor tweak that could help the agent from being stuck is randomizing the action order. The depth of the solution it will be totally random if the actions are randomized, so in this problem there is not a fixed path to solution through this search method.



```
Depth:  38713
Current node: No. 38714
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', 'C', '0', '0']
 ['0', '0', '😊', '0']

Depth:  38714
Current node: No. 38715
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', 'C', '0', '0']
 ['0', '😊', '0', '0']

Depth:  38715
Current node: No. 38716
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', 'C', '0', '0']
 ['😊', '0', '0', '0']

Depth:  38716
Current node: No. 38717
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', 'C', '0', '0']
 ['0', '😊', '0', '0']

Depth:  38717
Current node: No. 3871
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', '😊', '0', '0']
 ['0', 'C', '0', '0']

Depth:  38718
Current node: No. 3871
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', 'C', '0', '0']
 ['0', '😊', '0', '0'

Depth:  38719
Current node: No. 3872
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', '😊', '0', '0']
 ['0', 'C', '0', '0']

Depth:  38720
Current node: No. 3872
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['😊', 'B', '0', '0'
 ['0', 'C', '0', '0']
```

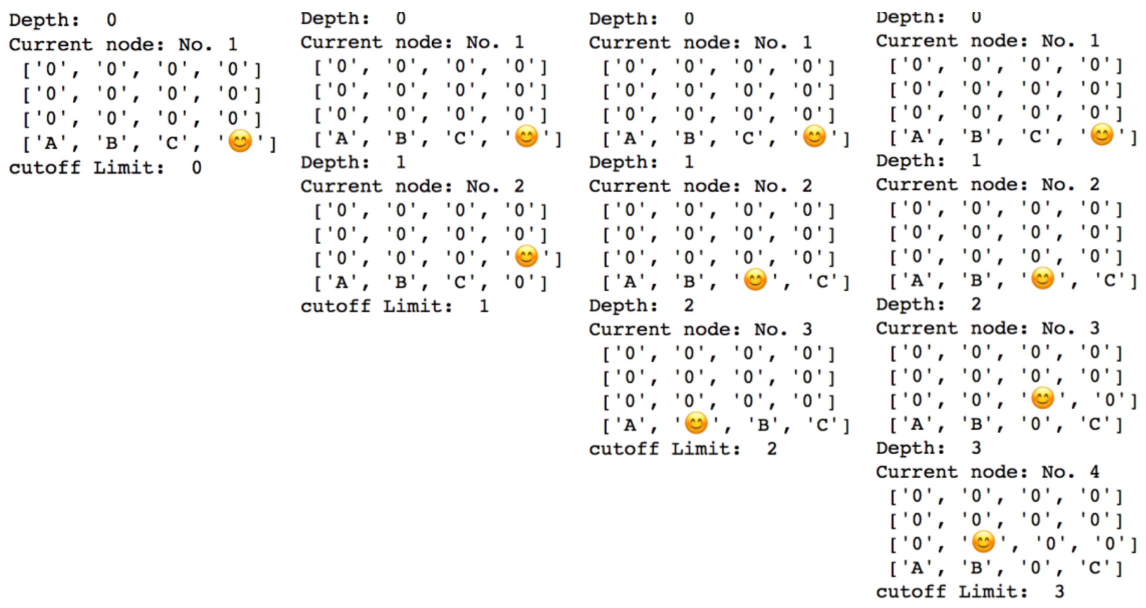
Figure 4. The last steps before finding a solution with a randomized action order.

In Figure 4, it is apparent that the path to the solution, from root to goal, is not the optimal in contrast to the BFS search.

ITERATIVE DEEPENING SEARCH

Iterative deepening search is a form of DFS but exploits the advantages of BFS. In the previous search, the solution was not optimal because other branches in the same depth were not being explored and if a solution existed in a swallower depth it would never be visited. The problem of the infinite branches is being solved with a depth limit that instantly acts as a dead end and forces the IDS explore other branches as BFS does.

After tweaking the action order in a randomized order to avoid being stuck. In this problem, the cutoff limit is the only way to force stop exploring a branch. In DFS before, a solution was found at depth 38720 and that solution came up only by luck. Since the actions are randomized it is possible enough that in limit $l + 1$ a solution can be found in the first branch in a depth d where $d \ll l$ and was not been found before.



```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
cutoff Limit:  0
```

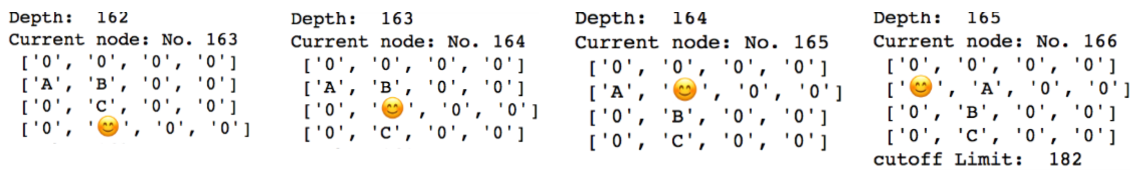
```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
Depth:  1
Current node: No. 2
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['A', 'B', 'C', '0']
cutoff Limit:  1
```

```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
Depth:  1
Current node: No. 2
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', '😊', 'C']
Depth:  2
Current node: No. 3
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', '😊', 'B', 'C']
cutoff Limit:  2
```

```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
Depth:  1
Current node: No. 2
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', '😊', 'C']
Depth:  2
Current node: No. 3
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '😊', '0']
 ['A', 'B', '0', 'C']
Depth:  3
Current node: No. 4
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '😊', '0', '0']
 ['A', 'B', '0', 'C']
cutoff Limit:  3
```

Figure 5. IDS search with randomized action order.

For every cutoff limit a different path is followed. In Figure 6, a solution was found in depth $d < l$ and you can see that it was found in the first branch. Iterative deepening search is both optimal and complete as it explores all the nodes in depth d where $d = l$. We cannot speak for optimality in this situation because every node is randomly generated and as bigger is the limit, the more probable is for a solution to be found in the shallowest depth.



```
Depth:  162
Current node: No. 163
 ['0', '0', '0', '0']
 ['A', 'B', '0', '0']
 ['0', 'C', '0', '0']
 ['0', '😊', '0', '0']
```

```
Depth:  163
Current node: No. 164
 ['0', '0', '0', '0']
 ['A', 'B', '0', '0']
 ['0', '😊', '0', '0']
 ['0', 'C', '0', '0']
```

```
Depth:  164
Current node: No. 165
 ['0', '0', '0', '0']
 ['A', '😊', '0', '0']
 ['0', 'B', '0', '0']
 ['0', 'C', '0', '0']
```

```
Depth:  165
Current node: No. 166
 ['0', '0', '0', '0']
 ['😊', 'A', '0', '0']
 ['0', 'B', '0', '0']
 ['0', 'C', '0', '0']
cutoff Limit:  182
```

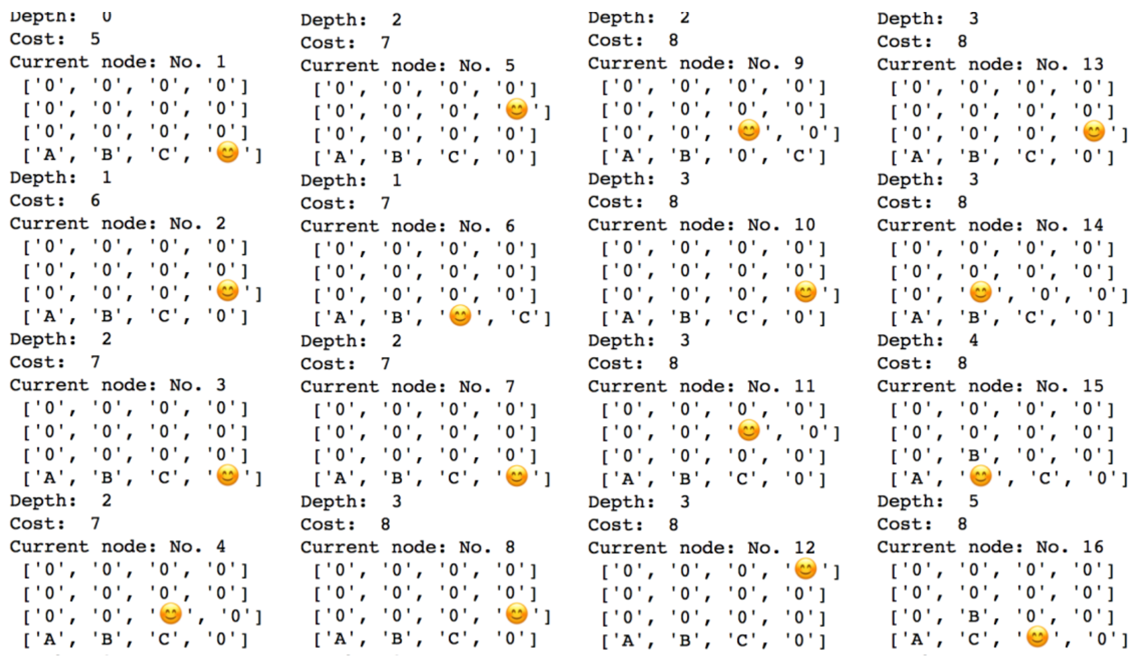
Figure 6. For every cutoff limit, random actions might give solution on the first branch.

A* SEARCH

The last algorithm is an informed search which uses a heuristic function to select the next node for expansion. As heuristic Manhattan distance was used. It is an admissible heuristic because the cost cannot be less from the actual cost of placing the tiles to their correct position, so it is not overestimating.

For every node visiting, add its successors to the frontier regardless if it will visit them. A* rearrange every time the frontier putting the nodes with the least estimated cost in front; if in the next steps finds out a better path from a different branch it will switch to that direction.

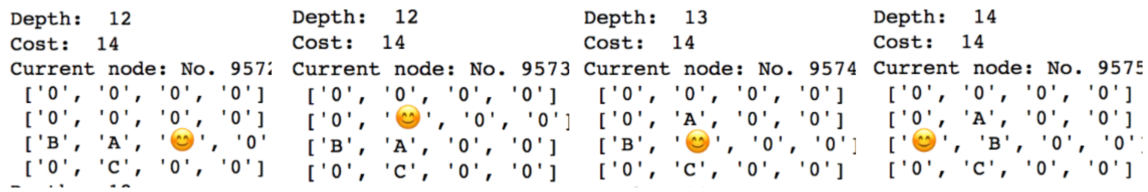
The root node has $h(n) = 5$, as the total distance of A, B, C tiles from their true position is 5.



```
Deptn:  0
Cost:  5
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
Depth:  1
Cost:  6
Current node: No. 2
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['A', 'B', 'C', '0']
Depth:  2
Cost:  7
Current node: No. 3
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
Depth:  2
Cost:  7
Current node: No. 4
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '😊', '0']
 ['A', 'B', 'C', '0']

Depth:  2
Cost:  7
Current node: No. 5
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
Depth:  1
Cost:  7
Current node: No. 6
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', '😊', 'C']
Depth:  2
Cost:  7
Current node: No. 7
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊']
Depth:  3
Cost:  8
Current node: No. 8
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['A', 'B', 'C', '0']

Depth:  2
Cost:  8
Current node: No. 9
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '😊', '0']
 ['A', 'B', '0', 'C']
Depth:  3
Cost:  8
Current node: No. 10
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['A', 'B', 'C', '0']
Depth:  3
Cost:  8
Current node: No. 11
 ['0', '0', '0', '0']
 ['0', '0', '😊', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
Depth:  3
Cost:  8
Current node: No. 12
 ['0', '0', '0', '😊']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']

Depth:  3
Cost:  8
Current node: No. 13
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊']
 ['A', 'B', 'C', '0']
Depth:  3
Cost:  8
Current node: No. 14
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '😊', '0', '0']
 ['A', 'B', 'C', '0']
Depth:  4
Cost:  8
Current node: No. 15
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', 'B', '0', '0']
 ['A', '😊', 'C', '0']
Depth:  5
Cost:  8
Current node: No. 16
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', 'B', '0', '0']
 ['A', 'C', '😊', '0']
```

Figure 7. A search tree in practice.*

In this specific starting position is very hard to find its way out, as almost every interaction with the tiles is going to worse the score. So, at first, it has to move in the blank tiles (increasing the score with every move) until its next interactions will start handling the score low.



```
Depth:  12
Cost:  14
Current node: No. 957
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['B', 'A', '😊', '0
 ['0', 'C', '0', '0']

Depth:  12
Cost:  14
Current node: No. 9573
 ['0', '0', '0', '0']
 ['0', '😊', '0', '0']
 ['B', 'A', '0', '0']
 ['0', 'C', '0', '0']

Depth:  13
Cost:  14
Current node: No. 9574
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['B', '😊', '0', '0']
 ['0', 'C', '0', '0']

Depth:  14
Cost:  14
Current node: No. 957
 ['0', '0', '0', '0']
 ['0', 'A', '0', '0']
 ['😊', 'B', '0', '0
 ['0', 'C', '0', '0']
```

Figure 8. The four last visited nodes.

It is worth noticing that at the start, almost every node being visited increased the score by one; in the end it got deeper by 3 levels keeping the score at 14.

Next, follows the total path from root to goal:

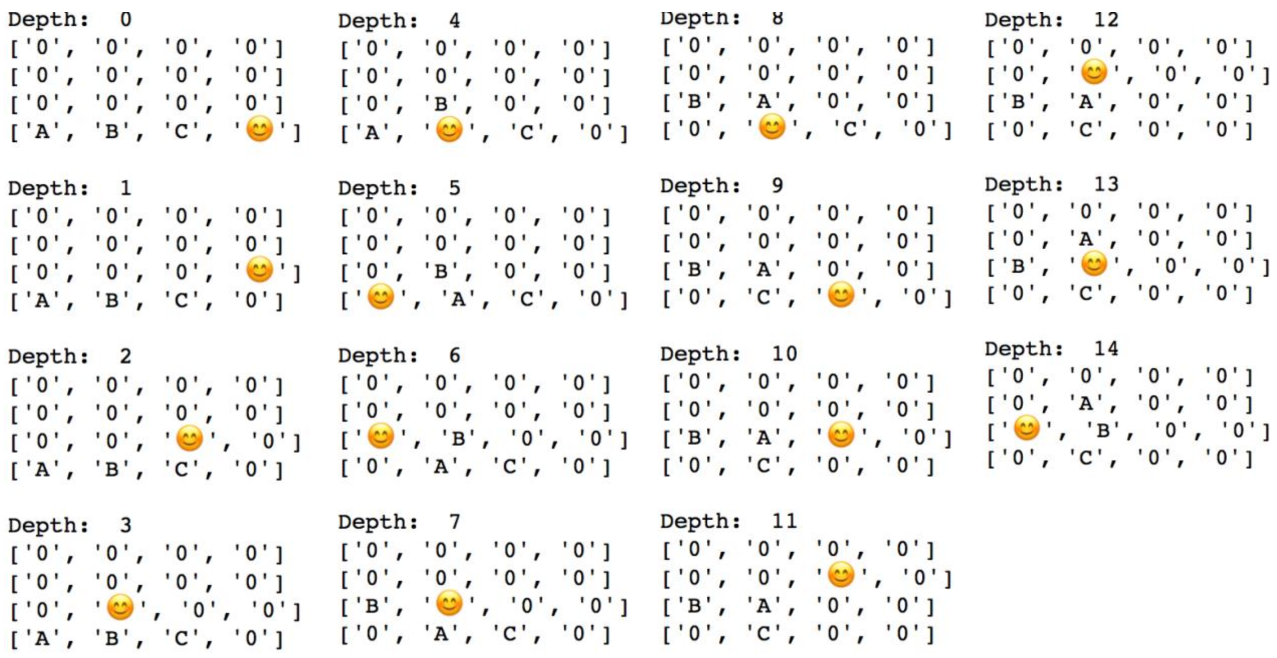

```
Depth:  0
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', '0', '0', '0']
['A', 'B', 'C', '😊']

Depth:  1
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', '0', '0', '😊']
['A', 'B', 'C', '0']

Depth:  2
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', '0', '😊', '0']
['A', 'B', 'C', '0']

Depth:  3
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', '😊', '0', '0']
['A', 'B', 'C', '0']

Depth:  4
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', 'B', '0', '0']
['A', '😊', 'C', '0']

Depth:  5
['0', '0', '0', '0']
['0', '0', '0', '0']
['0', 'B', '0', '0']
['😊', 'A', 'C', '0']

Depth:  6
['0', '0', '0', '0']
['0', '0', '0', '0']
['😊', 'B', '0', '0']
['0', 'A', 'C', '0']

Depth:  7
['0', '0', '0', '0']
['0', '0', '0', '0']
['B', '😊', '0', '0']
['0', 'A', 'C', '0']

Depth:  8
['0', '0', '0', '0']
['0', '0', '0', '0']
['B', 'A', '0', '0']
['0', '😊', 'C', '0']

Depth:  9
['0', '0', '0', '0']
['0', '0', '0', '0']
['B', 'A', '0', '0']
['0', 'C', '😊', '0']

Depth:  10
['0', '0', '0', '0']
['0', '0', '0', '0']
['B', 'A', '😊', '0']
['0', 'C', '0', '0']

Depth:  11
['0', '0', '0', '0']
['0', '0', '😊', '0']
['B', 'A', '0', '0']
['0', 'C', '0', '0']

Depth:  12
['0', '0', '0', '0']
['0', '😊', '0', '0']
['B', 'A', '0', '0']
['0', 'C', '0', '0']

Depth:  13
['0', '0', '0', '0']
['0', 'A', '0', '0']
['B', '😊', '0', '0']
['0', 'C', '0', '0']

Depth:  14
['0', '0', '0', '0']
['0', 'A', '0', '0']
['😊', 'B', '0', '0']
['0', 'C', '0', '0']
```

Figure 9. The solution is both optimal and complete.

SCALABILITY STUDY

To measure how these algorithms scale, time complexity was measured, that is the nodes generated until finding a solution. Of course, the action order will be randomized otherwise DFS and IDS will never reach a solution. The action order will be randomized too in BFS as it has no impact to the depth of the solution. It is already known from A* the optimal solution depth, but to make it more feasible the goal state will be modified to:

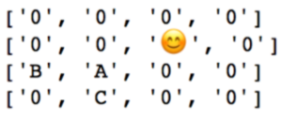

```
['0', '0', '0', '0']
['0', '0', '😊', '0']
['B', 'A', '0', '0']
['0', 'C', '0', '0']
```

Figure 10. Goal state for practical reasons. The ending state of agent also should be there.

Having this built in Python with bad use of data structures resulted in a space consuming over 7.5GB running in a whole night trying to find the original goal condition. With good use of C, a couple of minutes would be enough. That was the reason the goal state had to get modified. The average branching factor is 3[1], in BFS this would result in: $N_{BFS} = 1 + 3^1 + 3^2 + \cdots + 3^{14+1} - 3 \approx O(b^{d+1}) \approx 14\ million\ nodes$.

[1] 2 in four corners, 3 in 8 edges, 4 in 4 middle positions divided by 16 tiles of the board = 3.

```
Depth to goal state:  11 , Nodes generated until goal state:  274086
```

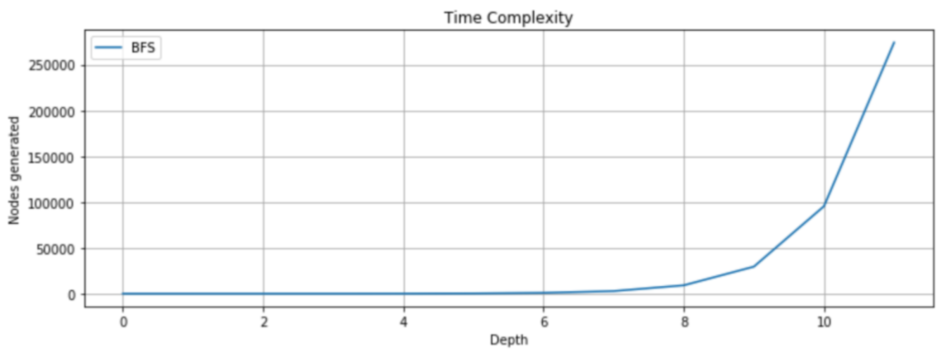


Figure 11. For d=11, it is estimated that $1 + 3 + 3^2 + 3^3 + \cdots + 3^{11} \approx 250k$ *nodes will be generated.*

Having the DFS tweaked, so the agent moves in a randomized order and not get stuck, the result is a linear generation of nodes as the search moves in the first branch it expands until it finds a solution, since the branch is infinite.

```
Depth to goal state:  16366 , Nodes generated until goal state:  16367
```

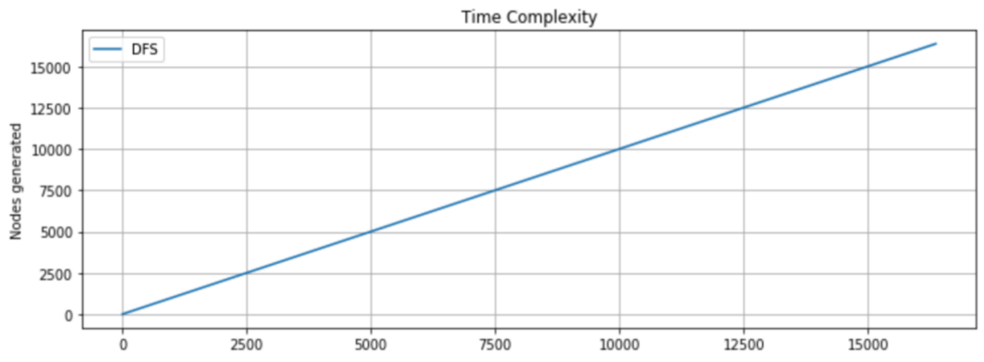


Figure 12. Nodes generated are d+1 because the root node count as a node in depth 0.

The IDS generates nodes continuously from depth 0 to depth limit until it finds a solution. We discussed already that when the limit gets bigger, it gets more probable to find a solution in depth $d < l$ in the first branch of the tree.

```
Depth to goal state:  34 , Nodes generated until goal state:  12125
```

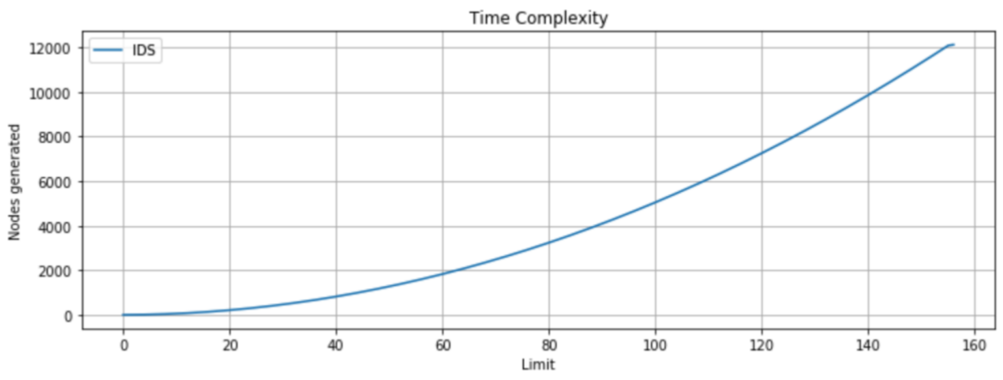


Figure 13. Nodes generated per limit in IDS.

In Figure 13 every limit carries all the generated nodes from previous limits. You can see that when the limit was 160 a solution was found in depth 34. In other words, from limit 159 to limit 160 only 34 nodes were created. A goal state in depth 34 trying to solve with BFS, would be catastrophic. Even the worst-case scenario of IDS with the same depth-limit would be slightly better.

A* needs to explore different number of nodes, depending on how many nodes have the same cost. It easily can be seen that in the depths 6-8 more nodes are accessed than in the very start or in the end.

```
Depth to goal state:  11 , Nodes generated until goal state:  7167
```

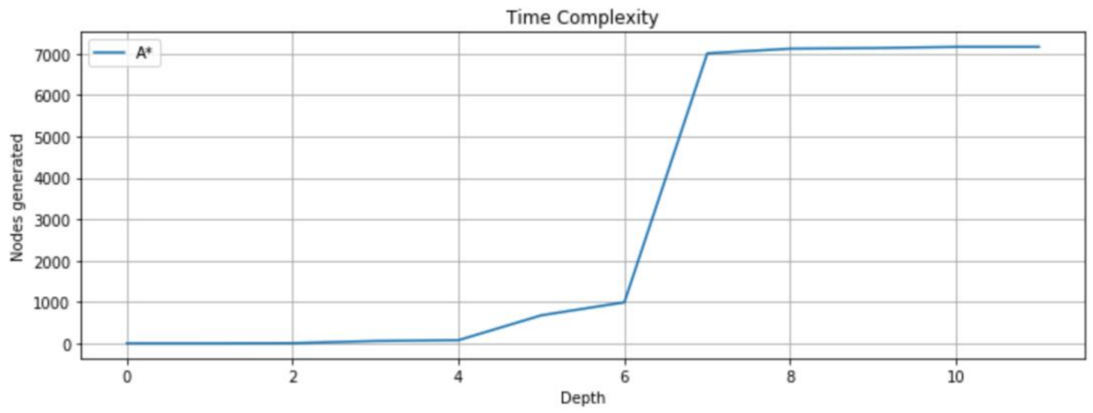
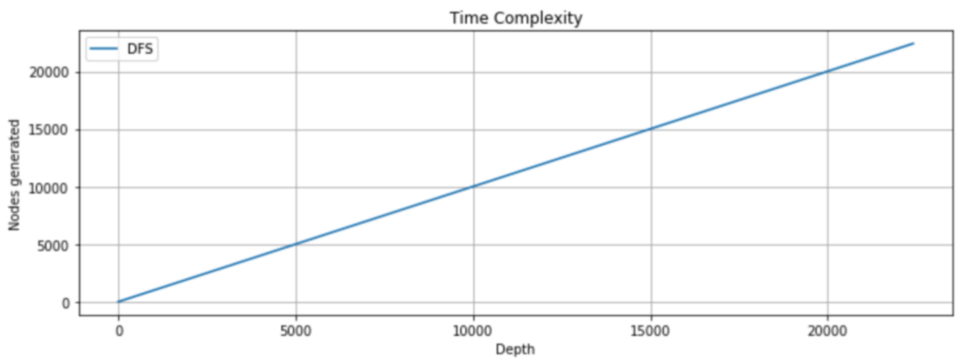


Figure 14. A access many more nodes in bigger depths than in the start.*

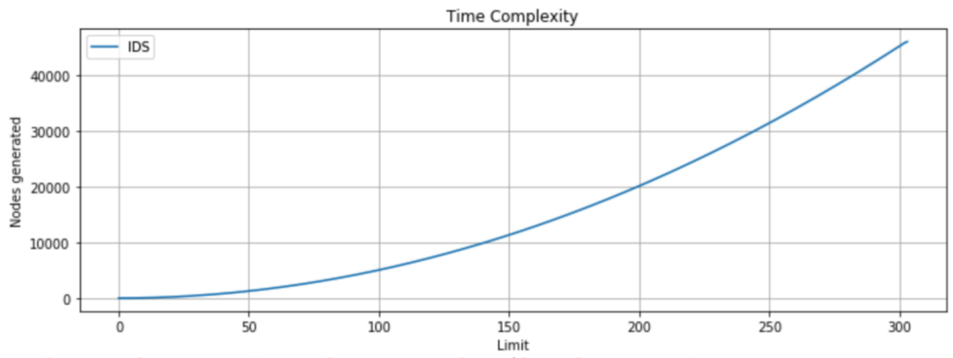
In Figure 14, you can see that A* explores significant less nodes than BFS having the same goal state.

The last diagrams show the time complexity for DFS, IDS and A* with the original goal condition.

```
Depth to goal state:  22411 , Nodes generated until goal state:  22412
```



```
Depth to goal state:  200 , Nodes generated until goal state:  45954
```



```
Depth to goal state:  14 , Nodes generated until goal state:  9575
```

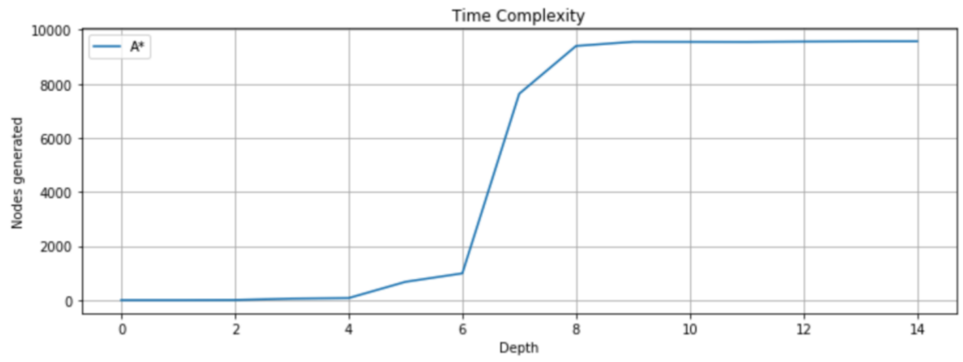


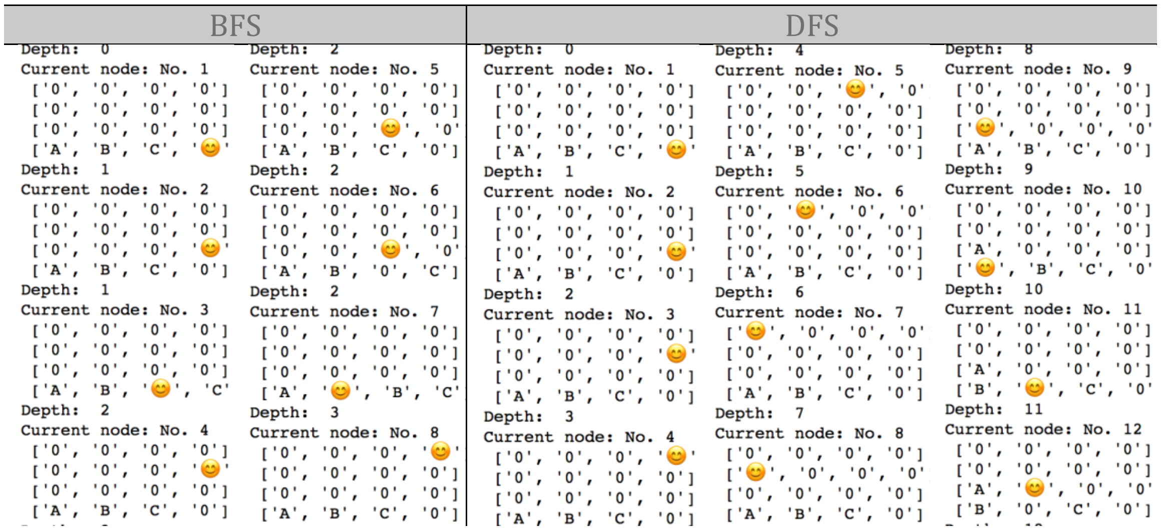
Figure 15. Accessed nodes for real goal are many more than the dummy goal before.

Comparing the 4 algorithms, anyone can see how much time consuming is BFS, but when it reaches to a solution, the solution will be optimal. The path from root to solution is eventually the same with A* algorithm. A* on the other hand reaches the same goal really fast but when the path score is the same for many nodes has to visit all the nodes.

DFS, goes to large depths really fast. It will never explore all the states just because the branches are infinite. However, IDS merges the benefits of BFS with DFS with less time complexity than BFS but in the specific problem it didn't work that way. For big depth limits, a solution will be found in the first branch as in the DFS.

EXTRAS AND LIMITATIONS

Investigating the standard problem, the advantages and disadvantages of the four search algorithms were shown in the form of time complexity. BFS complexity was so enormous that could not complete with the original goal condition. Another way to restrict the visiting nodes is to set a rule of not adding to the frontier duplicate nodes already visited (graph search). Of course, this strategy will have an impact to the space complexity as it has to store both visiting nodes and nodes in the frontier, but it will reduce significantly the time complexity.



BFS

```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊'
Depth:  1
Current node: No. 2
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊'
 ['A', 'B', 'C', '0']
Depth:  1
Current node: No. 3
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', '😊', 'C'
Depth:  2
Current node: No. 4
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊'
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  2
Current node: No. 5
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '😊', '0'
 ['A', 'B', 'C', '0']
Depth:  2
Current node: No. 6
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '😊', '0'
 ['A', 'B', '0', 'C']
Depth:  2
Current node: No. 7
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', '😊', 'B', 'C'
Depth:  3
Current node: No. 8
 ['0', '0', '0', '😊'
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

DFS

```
Depth:  0
Current node: No. 1
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '😊'
Depth:  1
Current node: No. 2
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊'
 ['A', 'B', 'C', '0']
Depth:  2
Current node: No. 3
 ['0', '0', '0', '0']
 ['0', '0', '0', '😊'
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
Depth:  3
Current node: No. 4
 ['0', '0', '0', '😊'
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  4
Current node: No. 5
 ['0', '0', '😊', '0'
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
Depth:  5
Current node: No. 6
 ['0', '😊', '0', '0'
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
Depth:  6
Current node: No. 7
 ['😊', '0', '0', '0'
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
Depth:  7
Current node: No. 8
 ['0', '0', '0', '0']
 ['😊', '0', '0', '0'
 ['0', '0', '0', '0']
 ['A', 'B', 'C', '0']
```

```
Depth:  8
Current node: No. 9
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['😊', '0', '0', '0'
 ['A', 'B', 'C', '0']
Depth:  9
Current node: No. 10
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', '0', '0', '0']
 ['😊', 'B', 'C', '0'
Depth:  10
Current node: No. 11
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', '0', '0', '0']
 ['B', '😊', 'C', '0'
Depth:  11
Current node: No. 12
 ['0', '0', '0', '0']
 ['0', '0', '0', '0']
 ['A', '😊', '0', '0'
 ['B', '0', 'C', '0']
```

Figure 16. Graphs of BFS, DFS. Notice that node No.5 in BFS has not the root state as a successor. Agent in DFS does not get stuck with ordered actions, from node No.8 it cannot go UP (visited), neither LEFT, so it moves DOWN.

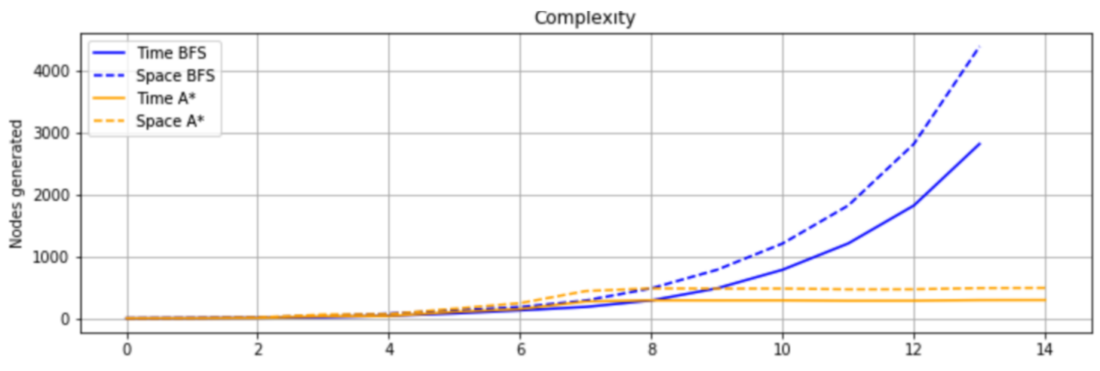


Figure 17. Comparison of complexity of BFS and A.*

```
Depth to goal state:  595 , Nodes generated until goal state:  603 , Nodes in memory:  1120
```

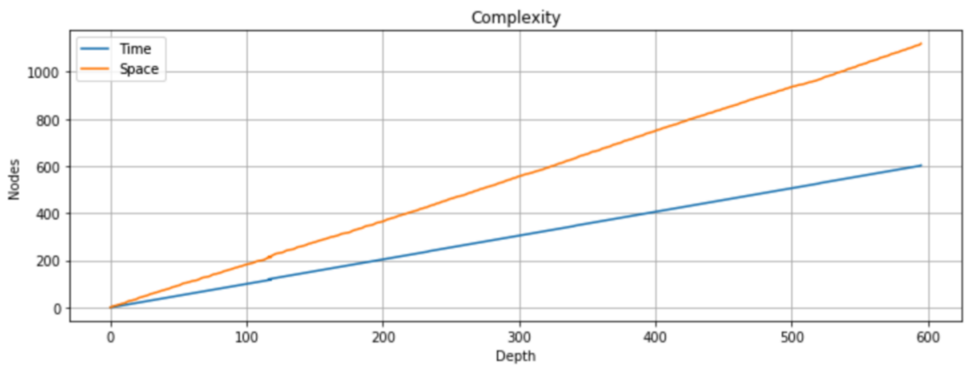


Figure 18. Complexity of BFS graph.

In DFS graph 603 nodes are generated from the root state while the depth is only 595, this means the starting branch was finite (all successors were visited or are already in frontier).

```
Depth to solution:  595 , Time complexity:  182106 , Space complexity:  1120
```

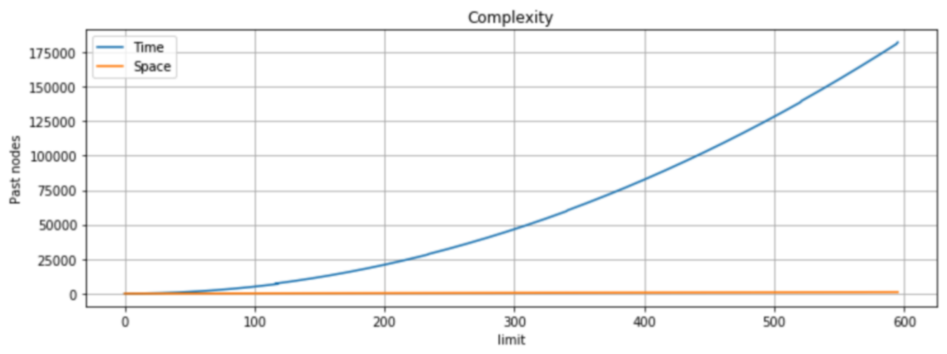


Figure 19. Complexity of IDS graph.

In a graph problem IDS is not very helpful as it has the same space complexity with DFS, but time complexity is enormous as it explores the same nodes continuously for every limit.

WEAKNESSES

As a weakness in the current approach, of course is my bad programming skill otherwise BFS goal condition would be feasible (it needs a lot of computations, but they are manageable). Also, the randomized action order does not seem the best way to study the problem as a) it is not reproducible, b) involves luck. Furthermore, could not come up with a way to study the worst-case scenario solution and lastly IDS moved to another branch only in graph (adjacent moves were unavailable) and that resulted in studying a more "complicated" DFS.