CS312 - Artificial Intelligence Lab
Assignment 1

Arun R Bhat (190010007), Mohammad Sameer (190010024)

1 Introduction

The goal of this assignment is to simulate in the state space Breadth-First Search (BFS), Depth-First Search (DFS), and Depth-First Iterative-Deepening (DFID) search. An *mxn* grid makes up the state-space. The initial position is (0,0). The position of (*) in the grid is the desired state. Moving UP, DOWN, LEFT and RIGHT are allowed (except for boundary).

We were assigned to compare the length of the path and the number of states using different search algorithms, and also to compare these between the orders in which neighbours are added.

2 Pseudo Code

In this section pseudo code of important function/algorithms are explained.

2.1 moveGen(state)

This function takes a state as input and outputs its valid neighbours that are either path or food.

Algorithm 1 moveGen(state)

```
neighbours ← list()                                        ▷ initialize neighbours to empty list
for neighbour of state in ORDER (DOWN, UP, RIGHT, LEFT) do
    if neighbour is NOT Boundary then
        neighbours.append(neighbour)
    end if
end for
return neighbours                                          ▷ neighbours of state are returned
```

2.2 goalTest(state)

This function takes a state and return True if it the goal/food, else it returns False.

Algorithm 2 goalTest(state)

```
if state.value = '*' then
    return True                                            ▷ Goal State reached
else
    return False                                           ▷ Goal State not reached
end if
```

2.3 BFS - Breadth First Search

Pseudo Code of the Breadth First Search Algorithm.

Algorithm 3 BFS(initialState, goalTest)

```
frontier ← Queue.new(initialState)
explored ← Set.new()
while frontier is NOT Empty do
    state = frontier.dequeue()
    explored.add(state)
    if goalTest(state) then
        return SolutionMaze(state)
    end if
    for neighbour in moveGen(state) do
        if neighbour not in frontier ∪ explored then
            frontier.enqueue(neighbour)
        end if
    end for
end while
return Failure
```

2.4 DFS - Depth First Search

Pseudo Code of the Depth First Search Algorithm.

Algorithm 4 DFS(initialState, goalTest)

```
frontier ← Stack.new(initialState)
explored ← Set.new()
while frontier is NOT Empty do
    state = frontier.pop()
    explored.add(state)
    if goalTest(state) then
        return SolutionMaze(state)
    end if
    for neighbour in moveGen(state) do
        if neighbour not in frontier ∪ explored then
            frontier.push(neighbour)
        end if
    end for
end while
return Failure
```

2.5 DFID - Depth First Iterative Deepening

Pseudo Code of the Depth First Iterative Deepening Algorithm.

Algorithm 5 DLS(state, depth, goalTest)

```
if goalTest(state) then
    return SolutionMaze(state)
end if
if state.distanceFromSource −1 = depth then
    return Failure
end if
for neighbour n in state.neighbours() do
    if n not Visited OR
        n.distanceFromSource > state.distanceFromSource + 1 then
        result ← DLS(nieghbour, depth, goalTest)
        if result then
            return result
        end if
    end if
end for
return Failure
```

Algorithm 6 DFID(initailState, goalTest)

```
for depth from 0 to ∞ do
    result ← DLS(initialState, depth, goalTest)
    if result != Failure then
        return result
    end if
end for
```

3 Directions to run code

The code `Group12.py` is added the zip file `Lab1_Group12.zip`. To run it use the following command:

```
$ python3 <codeName>.py <inputFile.txt>
```

This will create an `output.txt` file which will have the required output. Please refer to `readme.txt` in the zip file for more information.

4 Statistical and Graphical Analysis

The following tables and graphs record path lengths and number of states explored with varying maze sizes and order of neighbors added to analyze different algorithms.

Table 1 shows the number of states explored and path length with varying mazes sizes for different algorithms using the Order: `Down > Up > Right > Left`.

Algorithm	Order: Down >Up >Right >Left (Cell Width = 3, Cell height = 2)			
	Horizontal Cells	Vertical Cells	No. of States Explored	Path Length
BFS	2	2	15	10
DFS	2	2	11	10
DFID	2	2	80	10
BFS	4	4	42	24
DFS	4	4	51	26
DFID	4	4	842	24
BFS	6	6	113	34
DFS	6	6	47	40
DFID	6	6	4585	34
BFS	8	8	225	68
DFS	8	8	89	80
DFID	8	8	20864	68
BFS	10	10	327	94
DFS	10	10	278	114
DFID	10	10	40068	94

Table 1: Order: Down > Up > Right > Left

Table 2 shows the number of states explored and path length with varying mazes sizes for different algorithms using the Order: `Right > Left > Down > Up`.

Algorithm	Order: Right >Left >Down >Up (Cell Width = 3, Cell height = 2)			
	Horizontal Cells	Vertical Cells	No. of States Explored	Path Length
BFS	2	2	13	10
DFS	2	2	14	10
DFID	2	2	81	10
BFS	4	4	42	24
DFS	4	4	44	24
DFID	4	4	1048	24
BFS	6	6	111	34
DFS	6	6	115	34
DFID	6	6	3395	34
BFS	8	8	226	68
DFS	8	8	202	68
DFID	8	8	37643	68
BFS	10	10	325	94
DFS	10	10	225	94
DFID	10	10	98157	94

Table 2: Order: Right > Left > Down > Up

Figure 1 shows the how the number of states explored changes with varying mazes sizes for different algorithms using the Order: `Down > Up > Right > Left`.

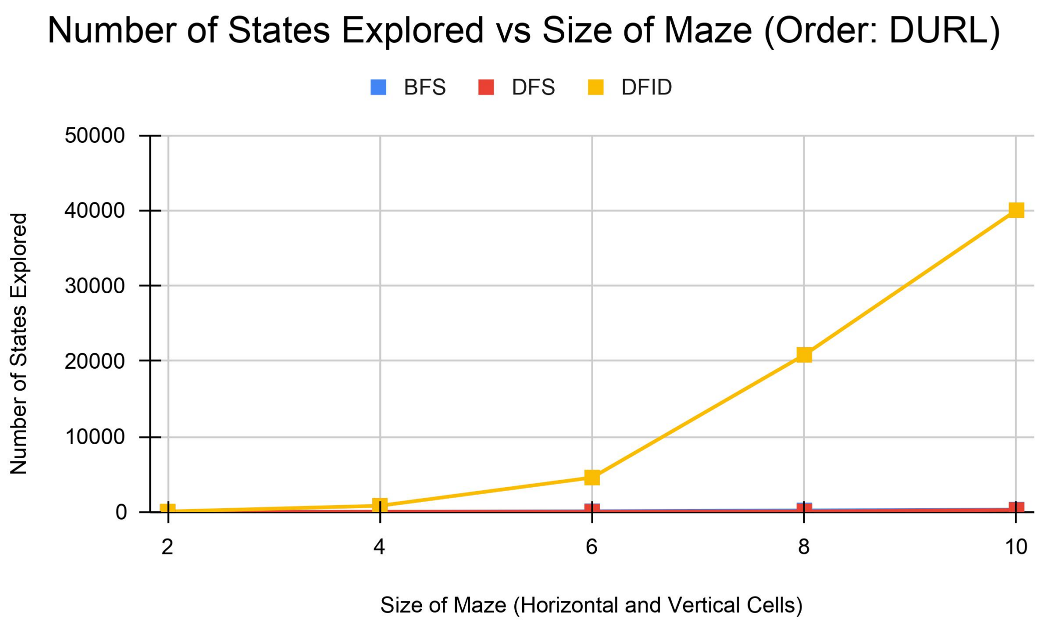

Figure 1: Order: Down > Up > Right > Left

Figure 2 shows the how the number of states explored changes with varying mazes sizes for different algorithms using the Order: `Right > Left > Down > Up`.

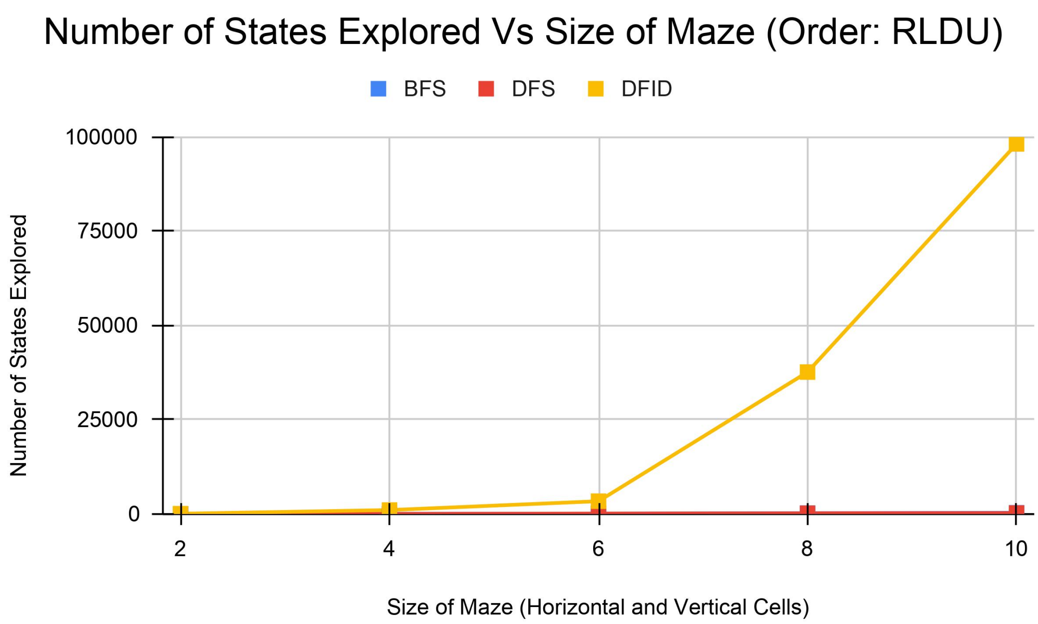

Figure 2: Order: Right > Left > Down > Up

Figure 3 shows the how the path length changes with varying mazes sizes for different algorithms using the Order: `Down > Up > Right > Left`.

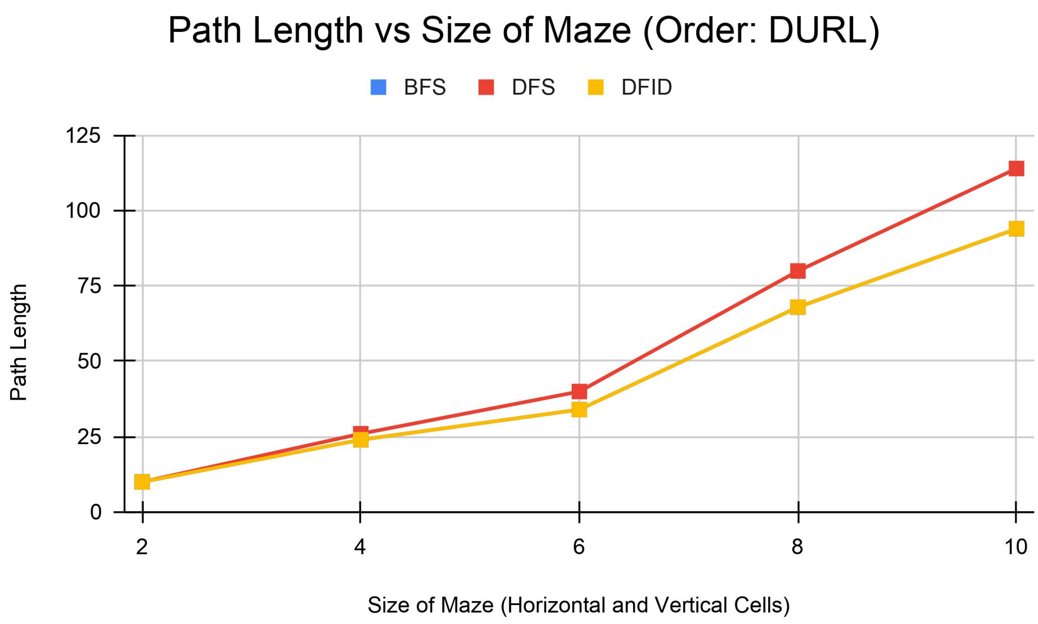

Figure 3: Order: Down > Up > Right > Left

Figure 4 shows the how the path length changes with varying mazes sizes for different algorithms using the Order: `Down > Up > Right > Left`.

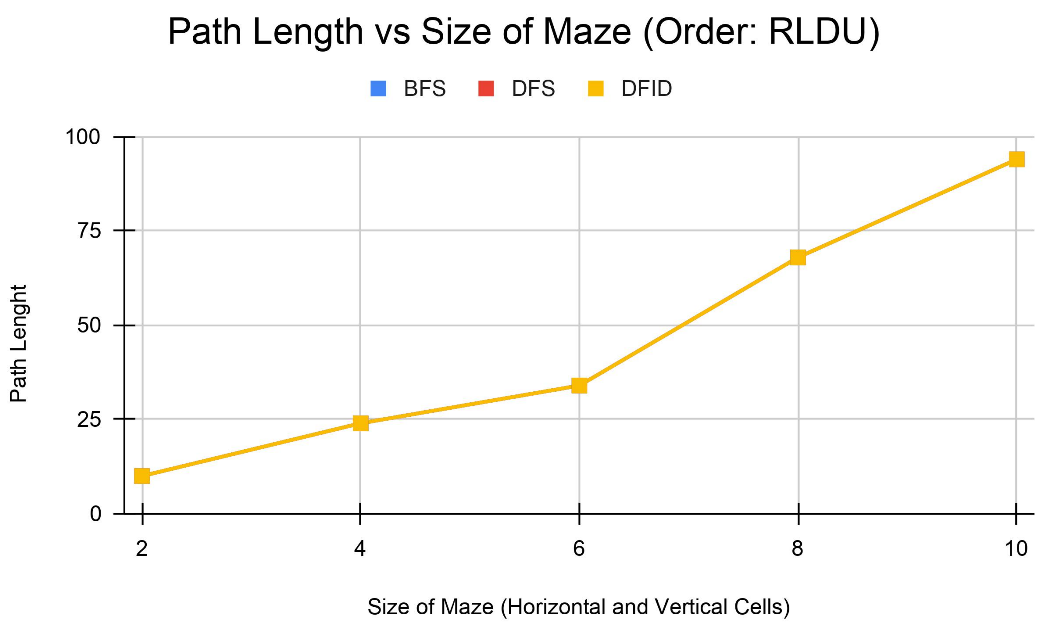

Figure 4: Order: Right > Left > Down > Up

5 Inference and Conclusion

- The path length obtained using BFS and DFID are more or less same. But DFS takes longer paths generally.
- The number of states explored by BFS and that by DFS do not differ significantly. But it is much larger when we use DFID algorithm. Since DFID tries to find the optimal path, it visits a node many times when traversing graph with cycles.
- The stats also prove that the number of states explored and the path length depend on order of traversal. But we can't always pick some order to give better results.
- DFID explores more number of states than others when the inputs are small. This can be explained by small branching factor and also the higher constant associated with the time complexity term.

The results of the dependence of number of states explored and the path length, given in the previous sections, are summarized in the table below.

Algorithm	**Dependence on order of Neighbours Added**	
	No. of States Explored	**Path Length**
BFS	True	False
DFS	True	True
DFID	True	False

Table 3: Dependence of order of Neighbours added