CS 314, Assignment 1 - Report

Shriram Ghadge (180010015), Rishit Saiya (180010027)

January 15, 2021

1 Abstract

The objective of this task is to simulate breadth-first search, depth-first search, and DFID in the state space. The state-space consists of an (m × n) grid. The start state is (0,0). The goal state is the position of [*] in the grid. The Pacman is allowed to move UP, DOWN, LEFT and RIGHT (except for boundary). A comparison of the path length and the number of states explored between the different search methods and, also between the orders in which neighbours are added, are performed.

2 Pseudo Codes

In the following subsections, pseudo codes for important functions in the code are explained.

2.1 movGen(state)

The function takes a state as input and returns a set of states that are reachable from the input state in one step. [Algorithm 1]

Algorithm 1 moveGen(state)

```
1: procedure moveGen(state)
2:     nextStates ← ()                                  ▷ initialize nextStates to empty set
3:     for neighbour n of state in order(DOWN,UP,RIGHT,LEFT) do
4:         if n is not boundary then
5:             nextStates.append(n)
6:     return nextStates                    ▷ nextStates are required moves generated
```

2.2 GoalTest(state)

This function returns True if the input state is goal and False otherwise. [Algorithm 2]

Algorithm 2 goalTest(state)

```
procedure GOALTEST(state)
    if $state.value$ == '*' then
        return $true$
    return $false$                                          ▷ state is not goal
```

2.3 BFS

This function is the implementation of BFS (Breadth First Search) Algorithm. [Algorithm 3]

Algorithm 3 final_BFS()

```
procedure FINAL_BFS
    if $Source$ && $goalCell$ == VALID then
        pushSource(queue)
    while !queue do
        statesExplored++;
        extractCell(queue);
        if $currentCell$ == $goalCell$ then                 ▷ Using Coordinates
            bool b = goalTest(goalCell);
            if $b$ == 1 then
                tracePath(source, currentCell);
                return $distance$
            pop(extractCell)
        movGen(state)
```

2.4 DFS

This function is the implementation of DFS (Depth First Search) Algorithm. [Algorithm 4]

2.5 DFID

This function is the implementation of DFID (Iterative Deepening Depth-First Search) Algorithm. [Algorithm 5]

With the code developed and some random customized Mazes were generated Delorie, it could be confirmed that our code was working fine. With `Cell Width = 3` and `Cell Height = 2` configuration, the values of path length and the number of states explored for each of the algorithm for each maze was noted at three different orders of visiting adjacent cells viz Down>Up>Right>Left, Down>Up>Left>Right, and Right>Down>Up>Left. An analysis of the results obtained is furnished below.

Algorithm 4 final_DFS()

```
procedure FINAL_DFS
    if $Source$ && $goalCell$ == VALID then
        pushSource(queue)
    while !stack do
        statesExplored++;
        extractCell(stack);
        if $currentCell$ == $goalCell$ then                ▷ Using Coordinates
            bool b = goalTest(goalCell);
            if $b$ == 1 then
                tracePath(source, currentCell);
                return $distance$
            pop(extractCell)
        movGen(state)
```

3 Data - Variation in Mazes

With variation in sizes of mazes, the corresponding data has been tabulated to further analyze different algorithms across various sizes of mazes.

The following ***Table 1*** shows the statistics related to mazes of different sizes created and corresponding number of states explored and path length across different algorithms using the Order: `Down > Up > Right > Left`.

The following ***Table 2*** shows the statistics related to mazes of different sizes created and corresponding number of states explored and path length across different algorithms using the Order: `Right > Left > Down > Up`.

4 Graphical Interpretation & Inference

4.1 Graphs

With the help of above data given in the tables, the following graphs are compared to analyse various components and efficiency of algorithms with varying sizes of mazes.

In ***Graph 1***, No. of States Explored vs. Size of Maze is plotted for the Order: `Down > Up > Right > Left`.

In ***Graph 2***, No. of States Explored vs. Size of Maze is plotted for the Order: `Right > Left > Down > Up`.

In ***Graph 3***, Path Length vs. Size of Maze is plotted for the Order: `Down > Up > Right > Left`.

Algorithm 5 final_DFID()

```
procedure FINAL_DFID
    if Source && goalCell == VALID then
        if currentDepth == 0 then
            return -1;
        distancefromSource(currentCell)
    if currentCell == goalCell then                    ▷ Using Coordinates
        bool b = goalTest(goalCell);
        if b == 1 then
            tracePath(source, currentCell);
            return distance
        statesExplored++;
    movGen(state)
    currentDepth = 1
    while TRUE do
        final_DFIS();
        if currentCell == goalCell then
            break
        currentDepth++;
```

In ***Graph 4***, Path Length vs. Size of Maze is plotted for the Order: `Right > Left > Down > Up`.

4.2 Inference

1. The number of states explored by BFS & DFS are generally different but do not vary substantially among themselves.
2. BFS and DFID yield paths of almost same length.
3. The number of states explored by DFID is relatively larger when compared to those explored by BFS and DFS due to the fact of finding optimal path in DFID with cycles within the graph may require visiting a node multiple times.
4. The order of visiting adjacent cells does affects the result obtained. It is evident from the plots of cumulative analysis seen above, though the difference in the path lengths obtained and number of states explored in each case is not substantially large.
5. BFS always yields paths of shorter (or equal to) lengths than those yielded by DFS.

Algorithm	**Order: Down > Up > Right > Left Cell Statistics (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	62	10
BFS	4	4	43	28
DFS	4	4	35	32
DFID	4	4	790	28
BFS	6	6	117	38
DFS	6	6	71	46
DFID	6	6	5183	38
BFS	8	8	137	88
DFS	8	8	194	102
DFID	8	8	7612	88
BFS	10	10	347	126
DFS	10	10	257	148
DFID	10	10	50437	126

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

5 Conclusion

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

Furthermore, for small inputs in DFID, it is observed that the increase in the number of explored states is due to the small branching factor and high constant attached with the time complexity. Whereas the BFS and DFS follow the conventional search pattern.

Algorithm	**Order: Right > Left > Down > Up Cell Statistics (Cell Width = 3, Cell Height = 2)**			
	Horizontal cells	**Vertical cells**	**No. of States Explored**	**Path Length**
BFS	2	2	13	10
DFS	2	2	13	10
DFID	2	2	63	10
BFS	4	4	41	28
DFS	4	4	48	28
DFID	4	4	1150	28
BFS	6	6	117	38
DFS	6	6	93	38
DFID	6	6	5648	38
BFS	8	8	137	88
DFS	8	8	184	88
DFID	8	8	20689	88
BFS	10	10	345	126
DFS	10	10	247	126
DFID	10	10	174250	126

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

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

Table 3: Dependence of order of Neighbours added

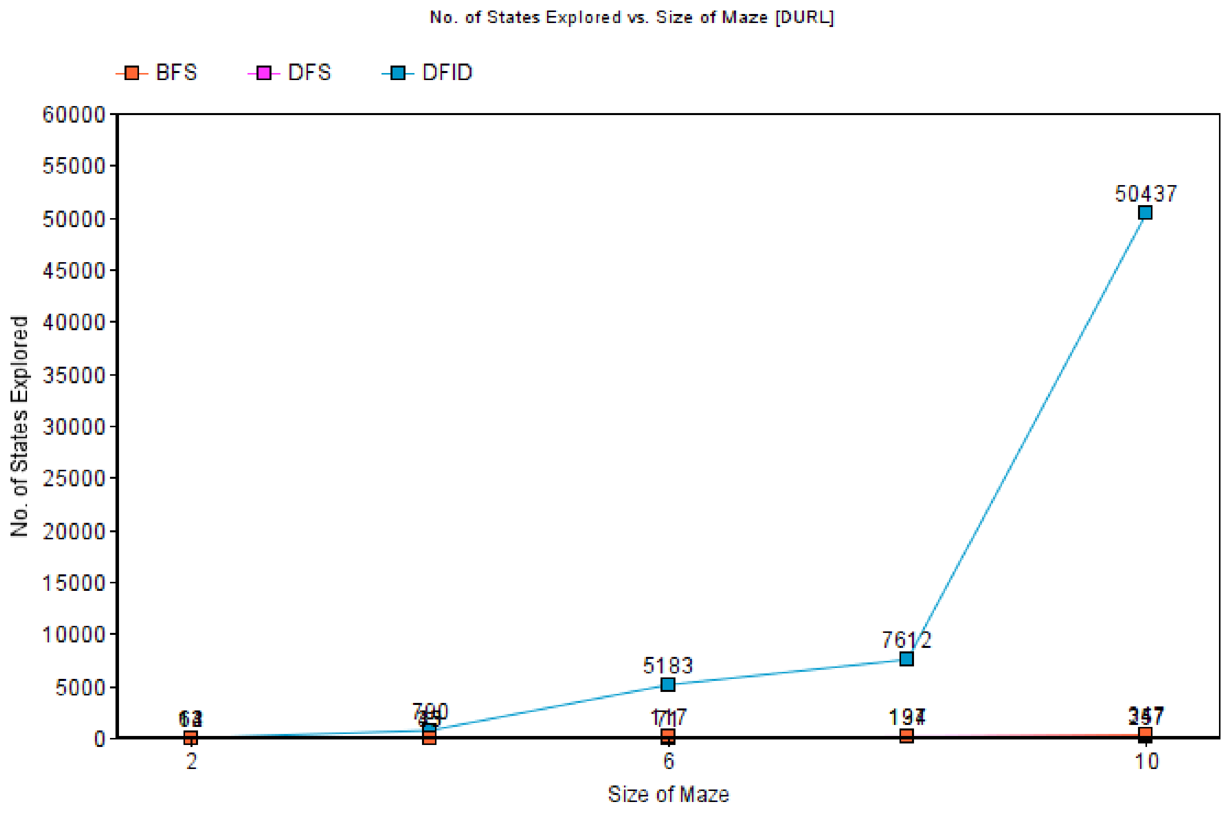

Figure 1: No. of States Explored vs. Size of Maze is plotted for the Order: `Down > Up > Right > Left`

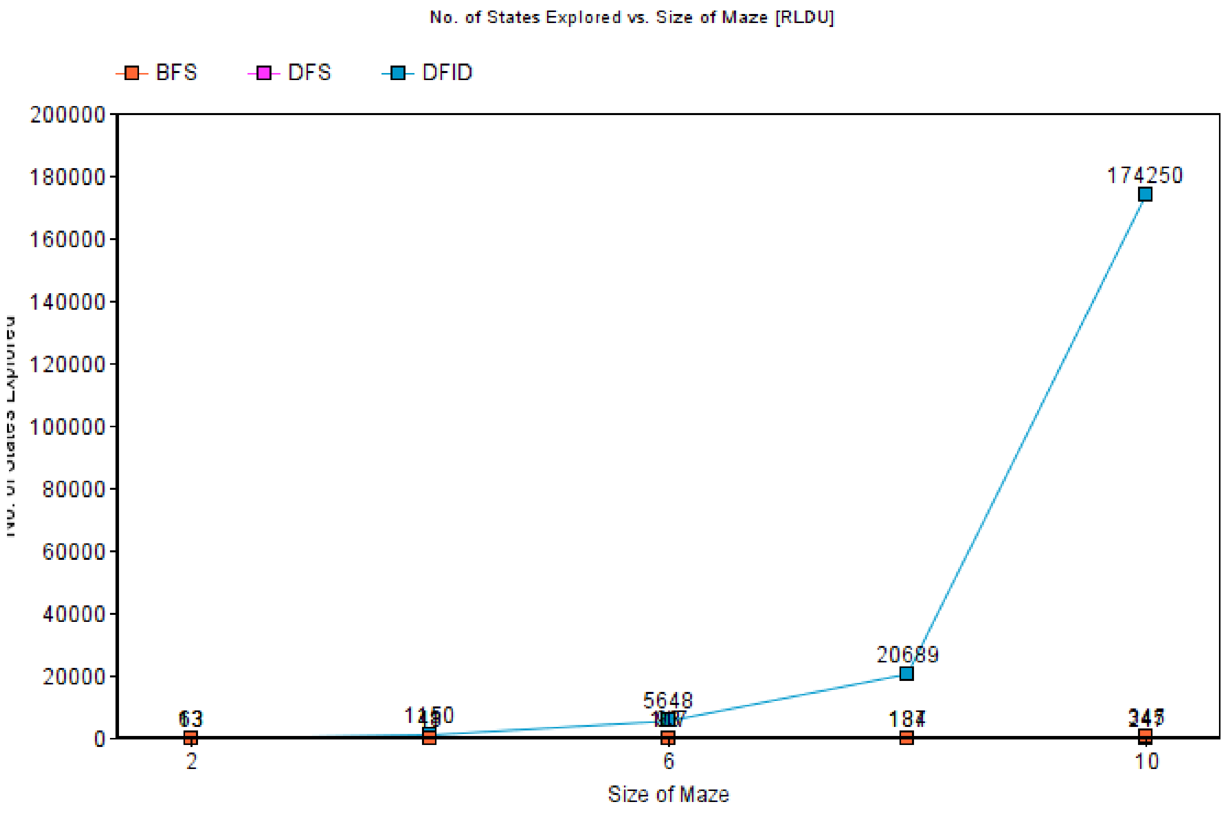

Figure 2: No. of States Explored vs. Size of Maze is plotted for the Order: `Right > Left > Down > Up`

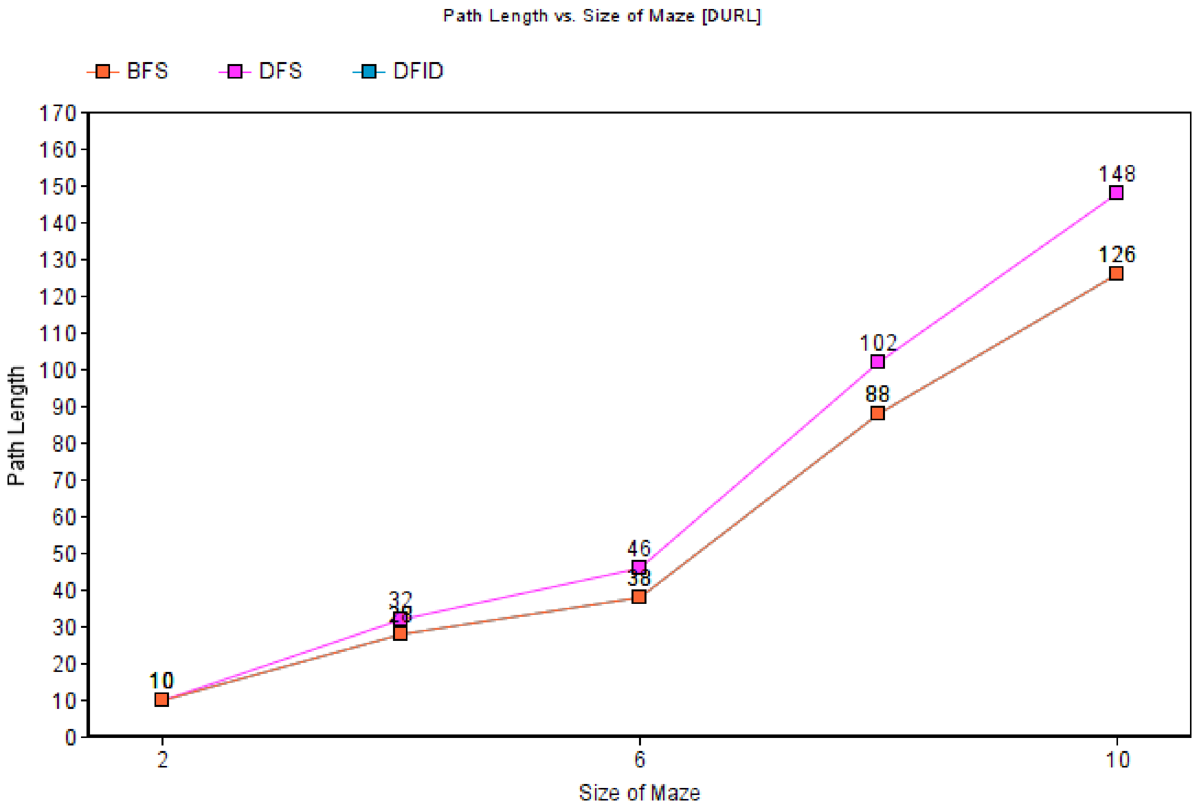

Figure 3: Path Length vs. Size of Maze is plotted for the Order: `Down > Up > Right > Left`

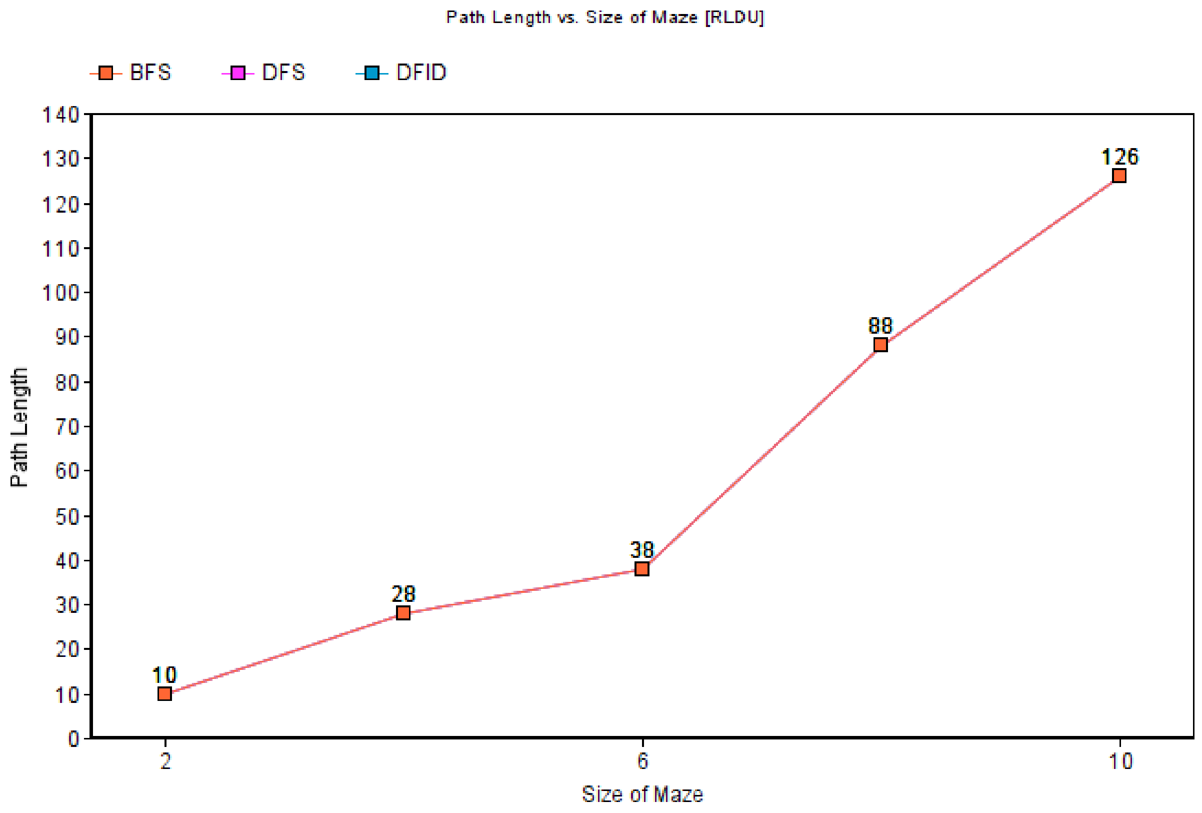

Figure 4: Path Length vs. Size of Maze is plotted for the Order: `Right > Left > Down > Up`