Assignment-1 Report

1. Introduction

In this assignment, I implemented different search algorithms to solve N-puzzle problem where N is 8. This a NP-hard problem and in the following sections their results are compared. The search algorithms are Breadth first search (BFS.py), depth first search(DFS.py), uniform cost search(UCS.py) and A*(Astar.py) search method.

2. Algorithms

- **Breath First Search :** Traverses the nodes at each level. Keeps a list of visited nodes. Frontier is a FIFO queue. O(b^{d+1}) worst case performance and memory allocation.

- **Depth First Search :** Expand deepest unexpended node. Fringe is a LIFO queue.

- **Uniform Cost Search:** Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. Uniform-cost search expands nodes according to their path costs form the root node. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost.

- **A star Search:** Avoids expanding paths that are already expansive. Uses heuristics in this case Manhattan distance. Keeps a list of visited nodes.

3. Implementation Details

- **Breath First Search:** Pseudo code an explanation

```
BFS (G, s)              //Where G is the graph and s is the source node
    let Q be queue.
    Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

    mark s as visited.
    while ( Q is not empty)
       //Removing that vertex from queue,whose neighbour will be visited now
       v  =  Q.dequeue( )

      //processing all the neighbours of v
      for all neighbours w of v in Graph G
          if w is not visited
                Q.enqueue( w )          //Stores w in Q to further visit its neighbour
```

mark w as visited.

- **Depth First Search:** Pseudo code an explanation

```
DFS-iterative (G, s):                    //Where G is graph and s is source vertex
    let S be stack
    S.push( s )          //Inserting s in stack
    mark s as visited.
    while ( S is not empty):
      //Pop a vertex from stack to visit next
      v  =  S.top( )
     S.pop( )
     //Push all the neighbours of v in stack that are not visited
    for all neighbours w of v in Graph G:
      if w is not visited :
            S.push( w )
           mark w as visited
```

- **Uniform Cost Search:** Pseudo code an explanation

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  if problem's initial state is a goal then return empty path to initial state
  frontier ← a priority queue ordered by pathCost, with a node for the initial state
  reached ← a table of {state: the best path that reached state}; initially empty
  solution ← failure
  while frontier is not empty and top(frontier) is cheaper than solution do
      parent ← pop(frontier)
      for child in successors(parent) do
          s ← child.state
          if s is not in reached or child is a cheaper path than reached[s] then
              reached[s] ← child
              add child to the frontier
              if child is a goal and is cheaper than solution then
                  solution = child
  return solution
```

- **A star Search:** Pseudo code an explanation

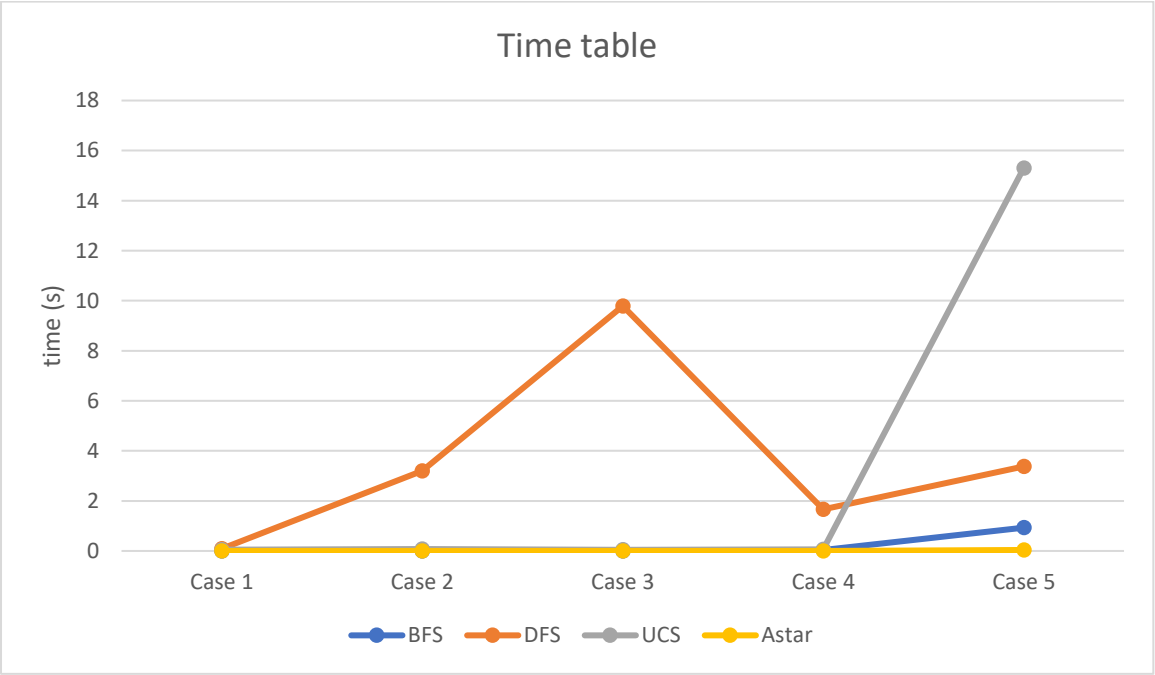
```
// Add the start node
put the startNode on the openList (leave it's f at zero)
// Loop until you find the end
while the openList is not empty
  // Get the current node
  let the currentNode equal the node with the least f value
  remove the currentNode from the openList
  add the currentNode to the closedList
  // Found the goal
  if currentNode is the goal
    Congratz! You've found the end! Backtrack to get path
  // Generate children
  let the children of the currentNode equal the adjacent nodes

  for each child in the children
    // Child is on the closedList
    if child is in the closedList
      continue to beginning of for loop
```

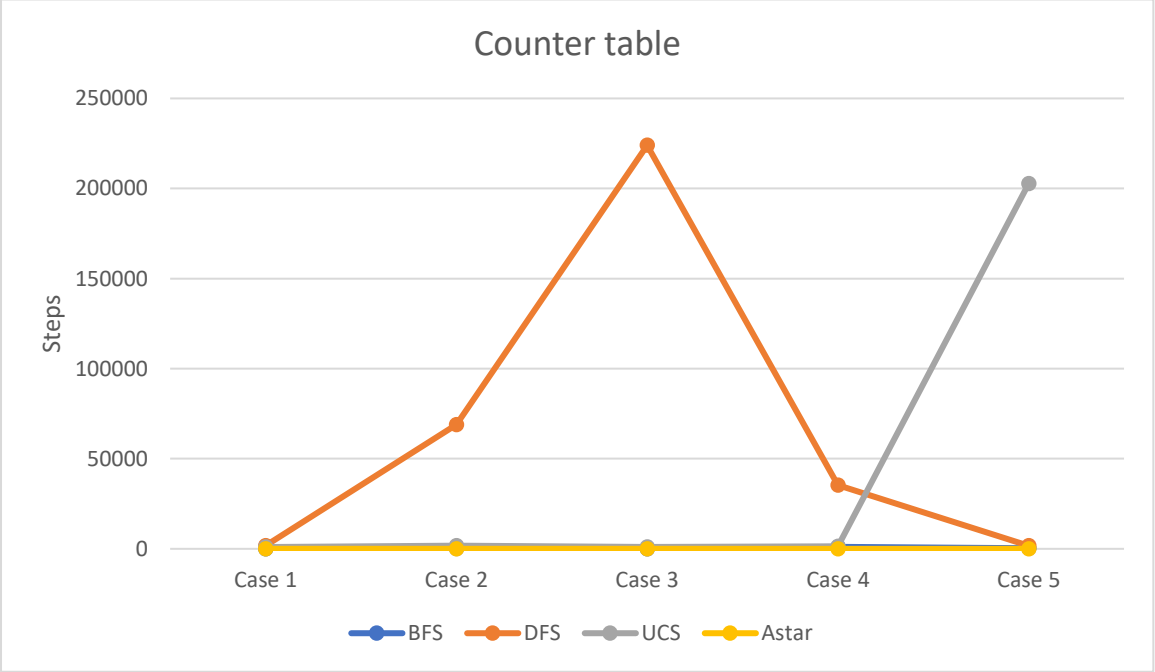
4. Results

Running the N-puzzle problem with different cases using the search algorithms leads to the following results

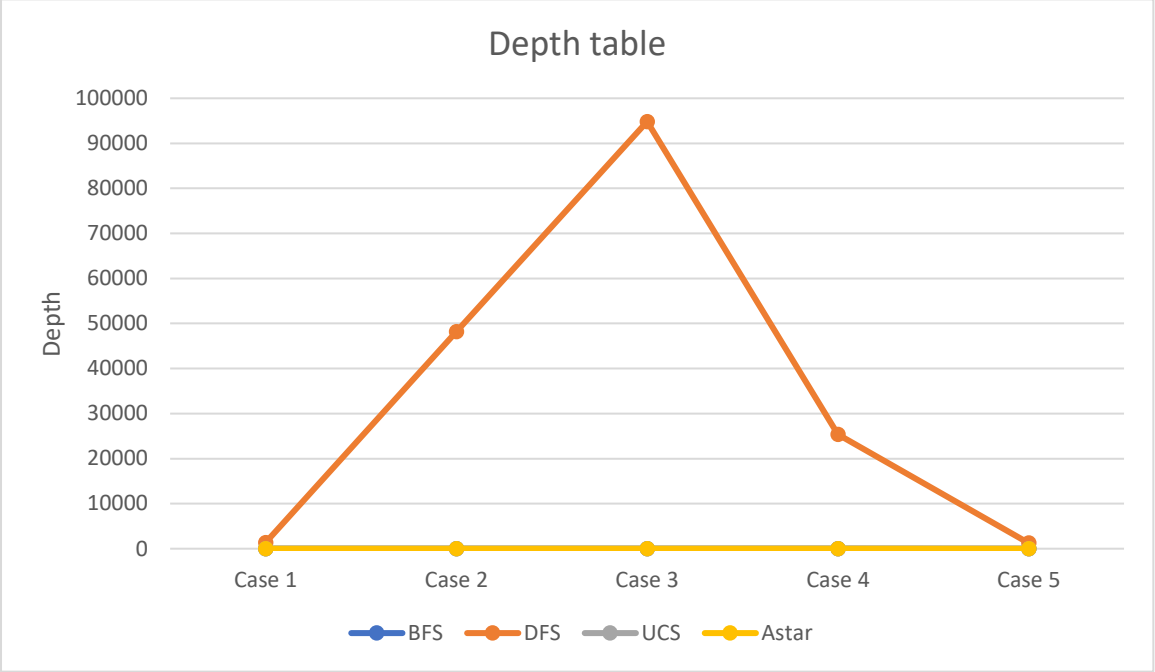
a) Time Comparison Table



b) Counter Table

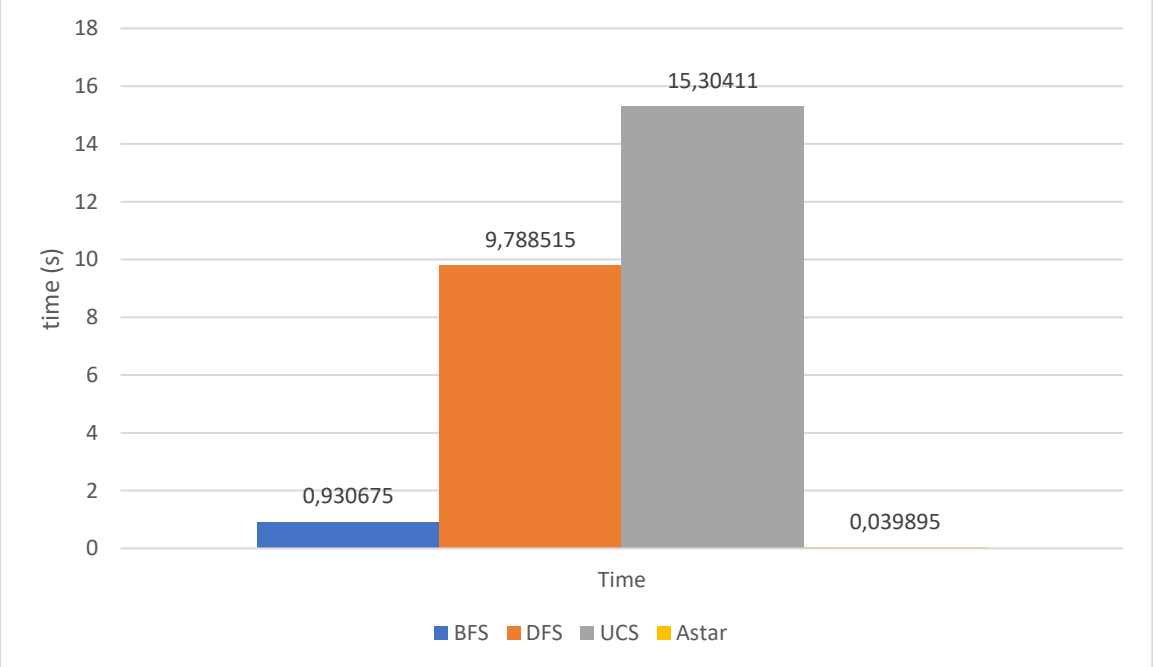


c) Depth Table

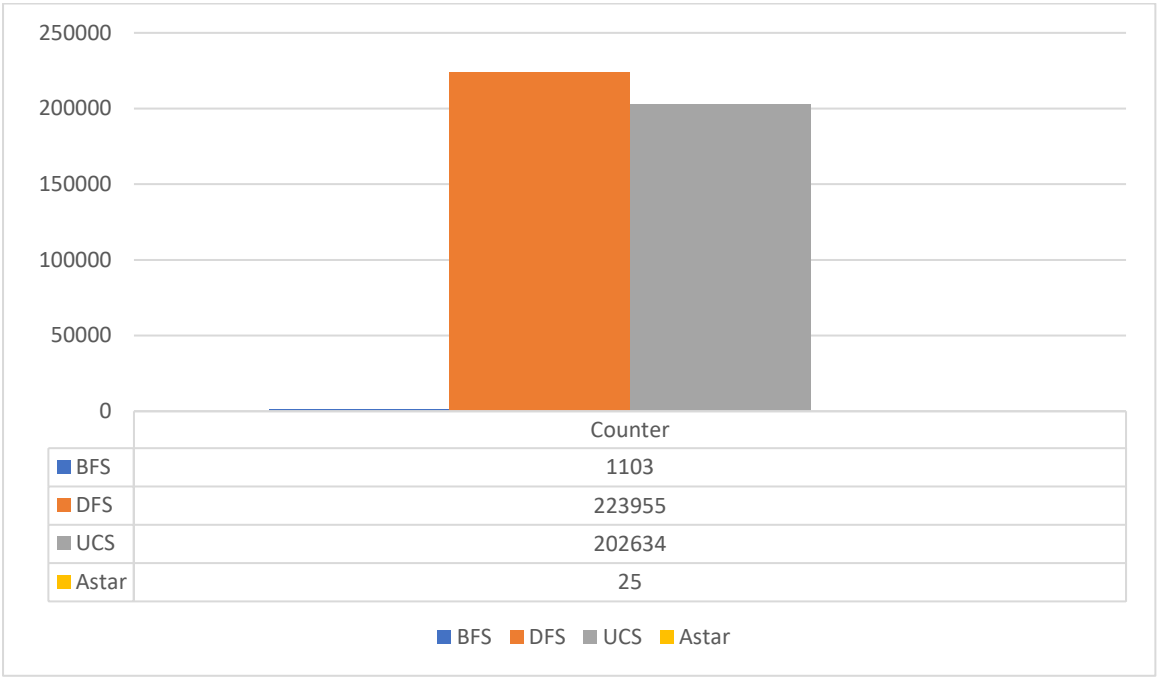


5. Conclusion

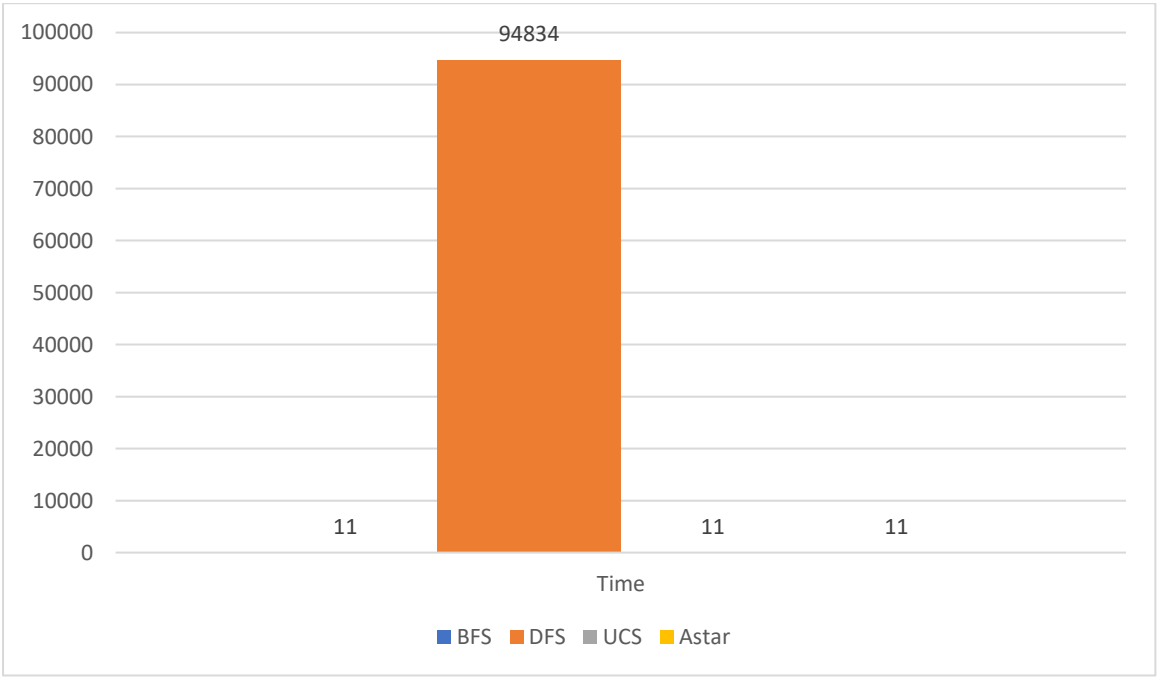
a) Worst-case Timetable



b) Worst-case Counter Table



c) Worst-case Depth Table



d) Discussion

DFS search gave the worst-case result in all cases.

UCS – A* always found the best solution