University of Dhaka

Department of Computer Science and Engineering

CSE-4111, Artificial Intelligence Lab

4th Year, 1st Semester

Session: 2019-20

Assignment No. 01

Title of the Assignment: Implementation and Performance Comparison of Classical Search Algorithms Using N-Puzzle Problem

Lab Group: B

Submitted by:

Meheraj Hossain (Roll - 24)

Date: 27th February, 2020

Submitted to:

1. Dr. Md. Mosaddek Khan, Assistant Professor, Dept. of CSE, DU
2. Dr. Muhammad Ibrahim, Lecturer, Dept. of CSE, DU

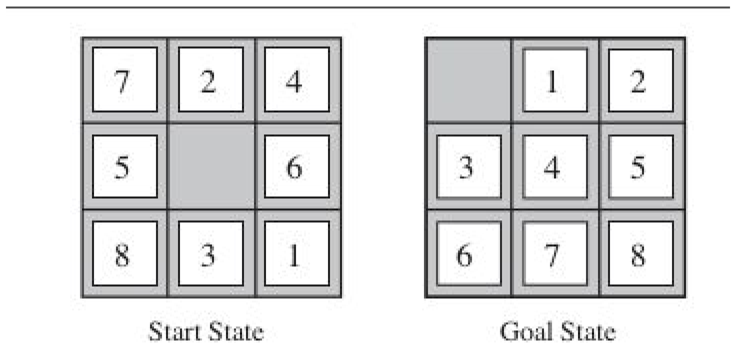
Problem definition:

In this assignment, I had to implement various classical search algorithms for solving n-puzzle problem. Besides, I had to analyze and the performance of these search algorithms for 8-puzzle problem and compare them by plotting in graph based on different performance metrics.

The algorithms are:

1. Breadth first search (BFS)
2. Uniform cost search (UCS)
3. Depth limited search (DLS)
4. Iterative deepening depth first search (IDS)
5. Greedy best first search (GBFS)
6. A* search

N-Puzzle is a well known problem where an nxn board configuration will be given and one can only move the blank cell to achieve the goal state. The actions that one can perform are UP, DOWN, RIGHT, and LEFT. The actions can only be applied if it is possible depending on the current state. Example:



A typical instance of the 8-puzzle.

Explanation :

1. **BFS:** According to the pseudo code given in the book.
2. **UCS:** According to the pseudo code given in the book.
3. **DLS:** According to the pseudo code given in the book and a limit of 26 is used.
4. **IDS:** As it is in the pseudo code of the book.
5. **GBFS:** Similar to UCS. In UCS the path cost of a node from the initial state is used as the evaluation function, f(n). On the contrary, in GBFS a heuristic function h(n) is used as f(n) i.e. f(n) = h(n).

6. **A*:** Implementation of A* is also similar to UCS and GBFS except the evaluation function. In A*, the evaluation function f(n) is the combination of both g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal i.e f(n) = h(n) + g(n). Here, h(n) is the heuristic function.

   The heuristics used for informed search algorithms(GBFS and A*) are:

   1. The number of misplaced tiles: Number of misplaced tiles between the current state and the goal state.
   2. The Manhattan distance: Summation of the Manhattan distances of each misplaced tile between the current state and the goal state. Manhattan distance of a misplaced tile is defined as:

      Manhattan distance = |X1 - X2| + |Y1 - Y2|

      Where (X1,Y1) and (X2, Y2) are the positions of the misplaced tiles.

   Note: Blank cell or tile was not considered as misplaced tile.

Result Analysis:

There are two parts in this assignment. They are:

1. Testing mode
2. Offline mode

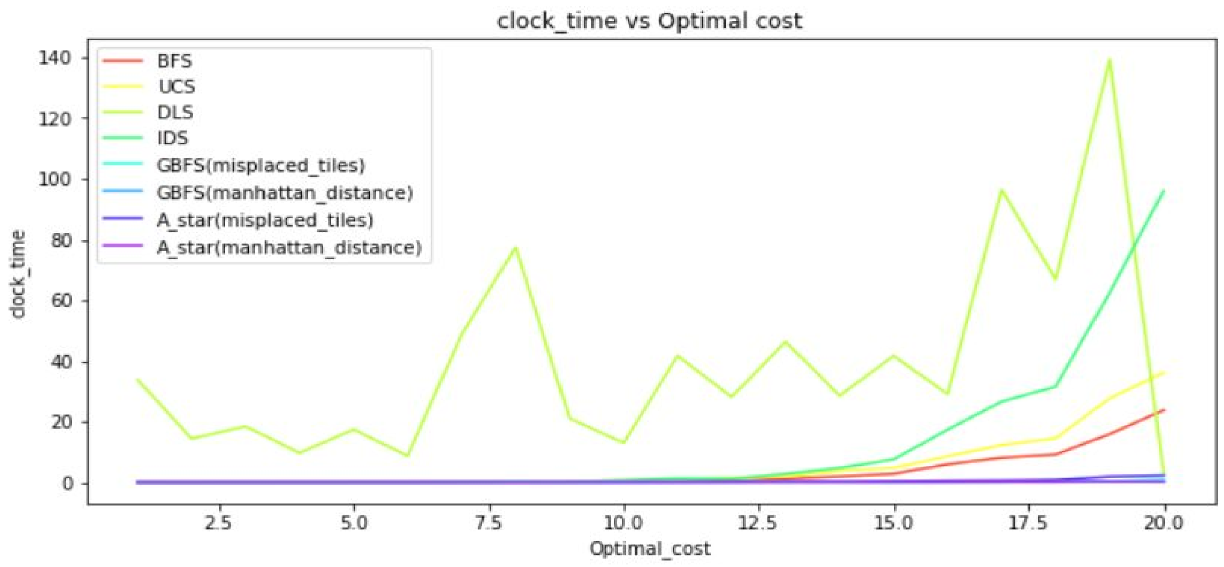
In testing mode, users will provide an initial board configuration and select a particular algorithm or all algorithms from the console. The outputs for the algorithms will be saved in separate text files for each algorithm.

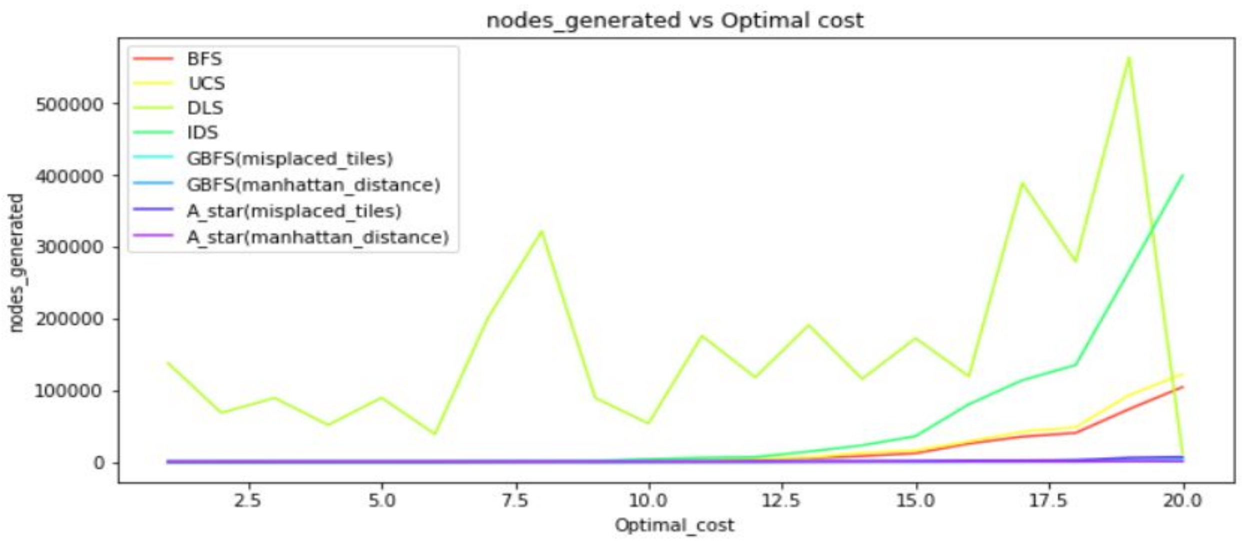
In offline mode, I took 20 input states of 8-puzzle problem where input states require varying steps, such as 1, 2, ..., 20 to reach the solution and ran all the six algorithms on these 20 inputs. Algorithms are compared using three performance metrics. They are:

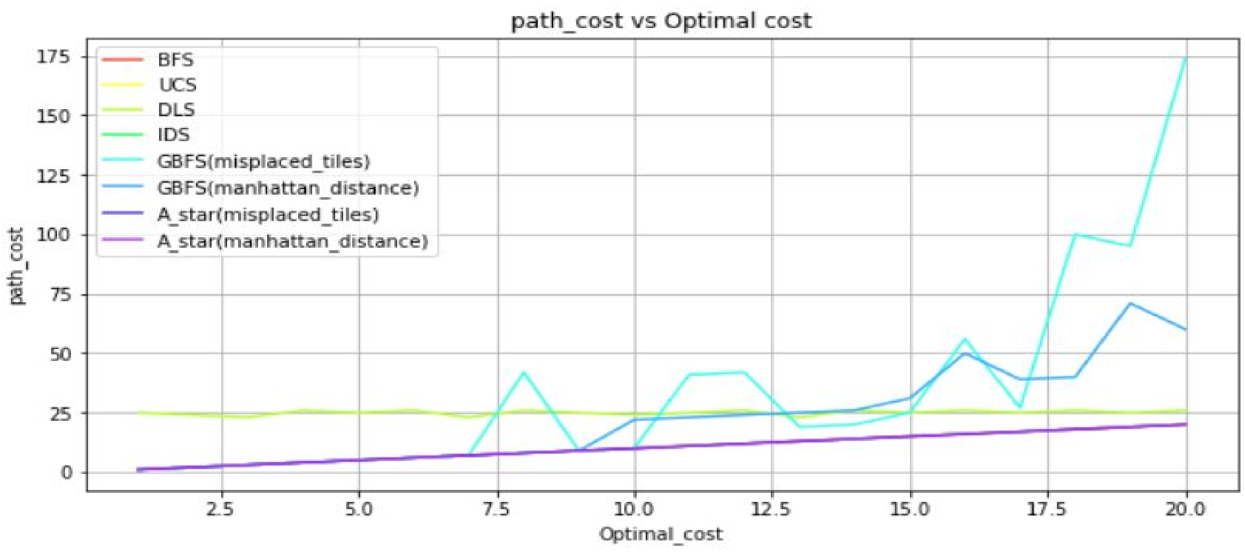
1. Clock time
2. Number of nodes generated
3. Path cost from initial state to goal state

The program generates three different plots for these three performance metrics and each plot contains 8 curves where 4 curves are for BFS, UCS, DLS, IDS and other 4 curves are for GBFS and A* with two different heuristics.

In the graph x axis represents optimal cost for the inputs and y axis represents performance metrics. The graphs generated by the offline mode is given below:


clock_time vs Optimal cost
clock_time
Optimal_cost
BFS
UCS
DLS
IDS
GBFS(misplaced_tiles)
GBFS(manhattan_distance)
A_star(misplaced_tiles)
A_star(manhattan_distance)
0
20
40
60
80
100
120
140
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


nodes_generated vs Optimal cost
nodes_generated
Optimal_cost
BFS
UCS
DLS
IDS
GBFS(misplaced_tiles)
GBFS(manhattan_distance)
A_star(misplaced_tiles)
A_star(manhattan_distance)
0
100000
200000
300000
400000
500000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


path_cost vs Optimal cost
path_cost
Optimal_cost
BFS
UCS
DLS
IDS
GBFS(misplaced_tiles)
GBFS(manhattan_distance)
A_star(misplaced_tiles)
A_star(manhattan_distance)
0
25
50
75
100
125
150
175
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Clock Time:

From “clock_time vs Optimal cost” graph, we can see that informed search algorithms i.e. GBFS and A* takes the minimum time as it uses a heuristic function to decide which node to be expanded next. In the case of BFS and UCS, clock time increases with increasing optimal depths.

Though in earlier cases IDS takes the same amount of time as BFS and UCS but increasing optimal depths results in a larger clock time. Because IDS iteratively increases its depth limit until it finds a solution. For each iteration IDS repeats the previous iteration again and again. So it actually does a lot of unnecessary iterations which increases the clock time.

From the graph, we can see that DLS takes the maximum time to find a solution within a specified depth limit. Also we can notice irregular spikes in the graph since DLS does not provide us with an optimal solution. As the depth limit was set to 26, DLS may explore a lot of unnecessary nodes/states and take a larger path to reach the goal.

Number of Nodes:

From the “nodes_generated vs Optimal cost” graph, we can see that this graph is almost similar to the “clock_time vs Optimal cost” graph. As clock time largely depends on the number of nodes generated during the execution of an algorithm, the logic behind this graph is also similar to the “clock_time” graph.

Path Cost:

From the “path_cost vs Optimal cost” graph, we can see that BFS, UCS, IDS and A* generates the same curve and in a y = x straight line form. Because these 4 algorithms provide us with a complete optimal solution. So, with increasing optimal depth solution path_cost increases linearly.

As GBFS and DLS do not provide optimal solutions, their curve does not follow any regular pattern. GBFS uses only heuristic as an evaluation function to decide which node will be expanded next. So, the solution of GBFS depends only on the heuristic and that’s the reason GBFS gives different path_cost for different optimal depth solutions. Path cost of GBFS increases in an irregular manner with the increasing optimal solution depths.

Path cost of DLS largely depends on the specified depth limit. Because DLS may explore a lot of unnecessary nodes/states and take a larger path to reach the goal.

Challenges Faced:

1. There is no pseudocode of GBFS and A* specified in the book. So It was a challenge for me to come up with an optimized GBFS and A* algorithm.

2. At first, I applied a backtracking approach in DLS and explored all possible paths which was a costly solution. But after a while I optimized DLS algo by expanding a node only if it is not visited before or if the limit for that node is greater than the previous time.