Artificial Intelligence

COSC 6368

Spring 2018

Final Project

Performance of Search Algorithms on N-puzzle

by

Bharat Surimalla (1526455)

Rahul Sawant (1632656)

Abstract

Search techniques and algorithms have been around us for a quite long time now. It is very important and interesting to know which algorithm performs better in which situation. Not all algorithms are equal and not all algorithms lead to final solution. Understanding of such search algorithms and using them for the right problem is the main intention of this project. To get better perspective of such algorithms we choose a n-puzzle problem and tried solving it using various known search algorithms.
A sliding block puzzle or commonly known as n-puzzle game provides an interesting foundation for search strategies in artificial intelligence. One can use various search strategies to solve n-puzzle problem using or without using heuristic algorithms.In this project, we covered a range of search techniques and its implementation in python. Our main intention is to find out the relations among different search algorithms and which one to chose over the other. This can be achieved by looking at the time and memory each search algorithm is using to solve the problem.

Keywords: DFS, BFS, A-Star, Search Algorithms, Solvability

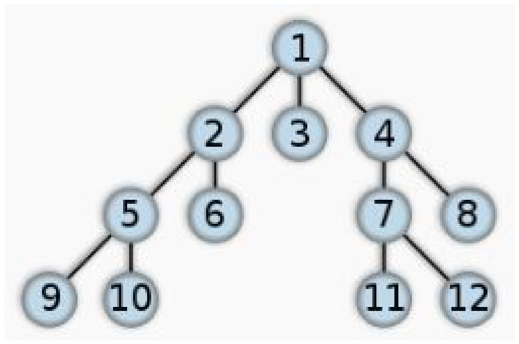
Search Algorithms

Can you imagine a day of your life without having to search for something, be it be car keys, books, pens, mobile charger etc. The same applies for computers, there is so much of data stored in it and when an user asks for some data it has to search through its storage and make it available to the end user. To make the searching fast and accurate there are many techniques or strategies which we are using in today’s world.
Different techniques give different performance results and accuracies. Some work better in some cases and some work better in other cases. Its is our duty and intuition to use the right search algorithm for the intended work we have.

Search Algorithms in Artificial Intelligence

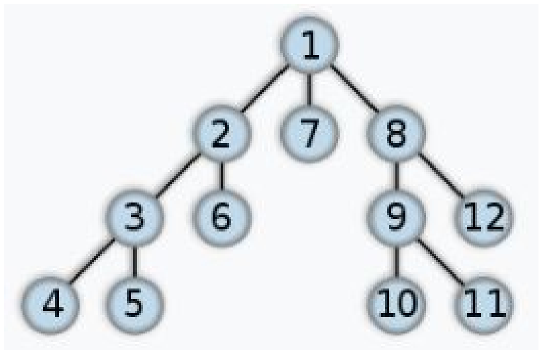
A simple algorithm do not specifically give the best possible solution as they are programmed to do a particular action for a particular state. Artificial Intelligence algorithms works towards a goal, identify the actions or series of actions that lead to the goal. It also considers impact of action on future states. Such type of actions give best possible solutions. In Artificial Intelligence, various search techniques are available with each having their own advantages and disadvantages. Here, we have implemented three search strategies to compare their performances on N-puzzle solver problem. Firstly, we will quickly go through these search techniques.

1. Breadth First Search



Breadth First Search was designed in late 1950s by *EF Moore*. This technique was invented to find optimal path out of a maze and discovered independently by *CY Lee* as a wire routing algorithm. Breadth-first search (BFS) is an algorithm for spanning or searching tree or graph data structures. It begins at the root node of a tree or some arbitrary node of a graph (sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbors. As we can see in the diagram, root node 1 does not have any neighbor, so it moves to next level neighbor and searches all existing nodes (here, node 2, 3 and 4) and then move to next level and it goes on until it reaches goal state.

2. Depth First Search



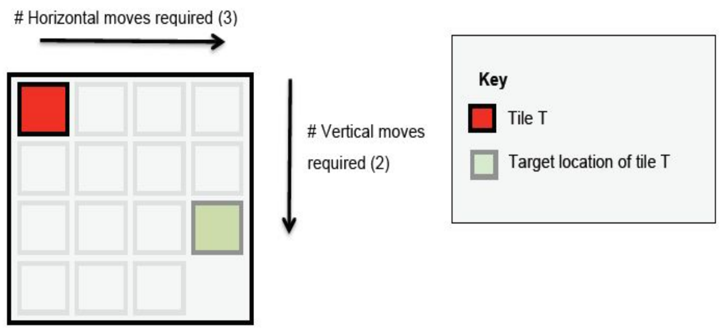
Another technique named depth-first search was explored by french mathematician *Charles Pierre Trémaux* in 19th century as a policy for solving mazes and come up with shortest path. Depth-first search (DFS), contrary to BFS, is a logic for traversing or searching tree or graph data structures by going deep inside the tree, unlike BFS where we first look around on the same level. This search technique begins at the root node (first node from the diagram) or by selecting some arbitrary node as the root in the case of a graph and explores as far as possible along each branch before backtracking. As we can see in the diagram, from root node 1, it goes all the way till level 4 neighbor, searching each node in way (here, 1-2-3-4) and then backtracking until it reaches goal state.

3. A* Search (using Manhattan Distance heuristic)

Peter Hart, Nils Nilsson and *Bertram Raphael* of Stanford Research Institute (now SRI International) first described the algorithm in 1968. It is an extension of *Edsger Dijkstra*'s 1959 algorithm. A* achieves better performance by using heuristics to guide its search. A* is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the solution (or goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node. Specifically, A* selects the path that minimizes,

$$f(n) = g(n) + h(n)$$

where n is the last node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal.

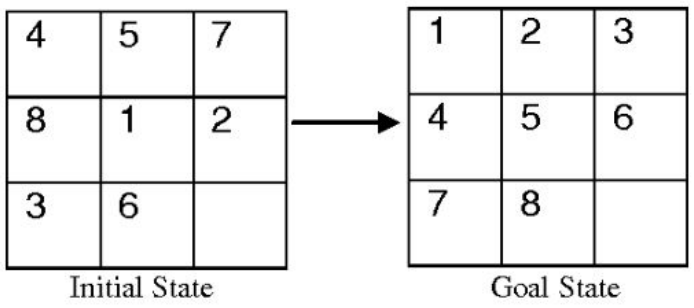

# Horizontal moves required (3)
# Vertical moves required (2)
Key
Tile T
Target location of tile T

Moreover, The number of moves needed to solve any model can be calculated quickly and efficiently using a relatively simple piece of code which sums up the minimum adjacent moves required to move every tile into its target location. This algorithm can be used as an admissible heuristic for the N-Puzzle, and is called the Manhattan Distance. The Manhattan distance for situation demonstrated is 5. It is actually a distance between two points measured along axes at right angles. Limitation of the Manhattan Distance heuristic is it considers each tile independently, but in real tiles interfere with each other.

Implementation of Search Algorithms on N-puzzle

- **N-puzzle game**

This is a very well known game which consists a square board with √(n+1) rows and columns. This is filled with n square tiles such that it has a gap on the board to slide the squares to adjacent positions. Each tile is marked with a number on it. The aim of the game is to transform the initial arrangement to target arrangement with a series of valid moves. This simple game makes a very fascinating platform for the artificial Intelligence for number of reasons.This might look very simple on the face of it but the actual toughness of it comes from the number of possible permutations that each n-puzzle game has.



- **Problem Solving**

Given a specific board configuration our aim is to find the shortest route possible to reach target board configuration using one of the three search algorithms from users choice.
We broke down the project into two parts for our convenience and they are as follows.

1. Search Strategy
2. Solvability

1. **Search Strategy**

For implementing BFS algorithm, we can start with ‘deque’ as a queue, or even a simple list in python. Eventually, using list will make algorithm slower. We assign initial node into the queue and then repeat this procedure until we visit the goal node or all available nodes: take the first from the queue, check if it was visited or not, check if it's the goal, put all neighbors at the end of the queue, repeat. For each step, we track not only the nodes but directions and the path for the current node too. BFS is optimal and is guaranteed to find the best solution that exists but major drawback is time complexity is O(|V|+|E|), where |V| is a number of nodes and |E| is a number of edges in the graph.
Depth-first search (DFS) is an algorithm similar to BFS. In DFS non-recursive implementation, we are using a stack instead of a queue to store nodes which will be exploring. This way we check the nodes

first which were last added to the stack. DFS is not optimal and it is not guaranteed to find the best solution. Hence, it is not a good choice to find a optimal path in a maze. Although, it has other applications in finding connected components or maze generation.

A* is a widely used pathfinding algorithm and an extension of Edsger Dijkstra algorithm. For A* we take the first node which has the lowest sum path cost and expected remaining cost. But heuristics must be admissible, that is, it must not overestimate the distance to the goal. It uses a heuristic cost function of node to determine the order in which the search visits nodes in the graph. For a maze, one of the most simple heuristics can be "Manhattan distance". The time complexity of A* depends on the heuristic.

2. Solvability

Our initial aim is to find out the efficient computational methods to solve n-puzzle game. We set out with the intent of exploring various search algorithms like bfs, dfs, A-star and found out that A-star is most fitting strategy.

Not all N-puzzle problems are solvable, which is a very important thing to know before proceeding with the solution. Looking for the solution of an unsolvable problem might keep on running forever and take up all the computational power.This is first crucial step to figure out if it has a solution or not.

There are 3 fundamental things that need to considered when looking for the solvability of a n-puzzle problem.

1.position of tiles.

2.position of the blank space.

3. Dimension of the n-puzzle Board.

Rules for a given n-puzzle game to be solvable.

>> If the grid width is odd, then the number of inversions needed is even in a solvable situation.

>> If the grid width is even and the blank is on even row counting from the bottom then the number of inversions needed is odd in a solvable situation.

>> If the grid width is even and the blank is on odd row counting from the bottom then the number of inversions needed is even in a solvable situation.

We verified the above results using the following.

>> The parity of the dimension.

>> The parity of the number of the row in which the space is located.

>> The parity of the number of inversions.

Result of Research

Now, we will look at the results of 3 search algorithms mentioned above. There are many parameters which differentiate these search algorithms. We know that A* search algorithm is optimal and requires less time to find solution, but it is not always true.

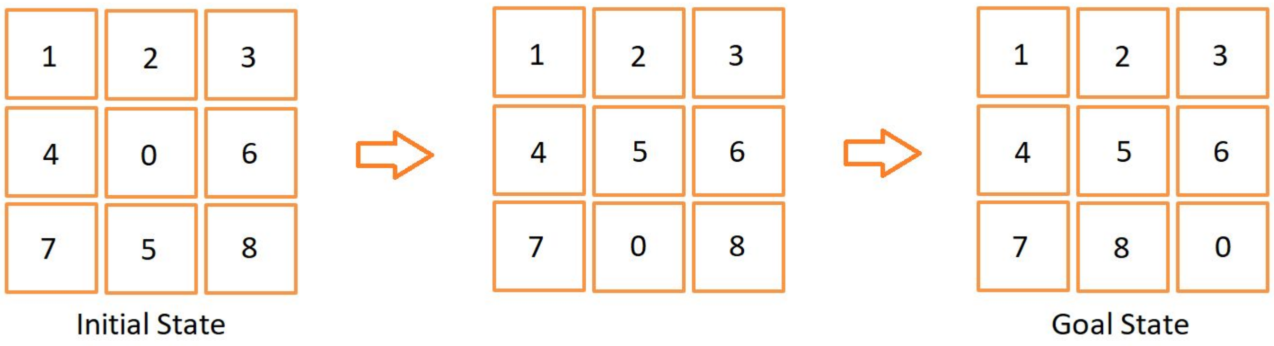
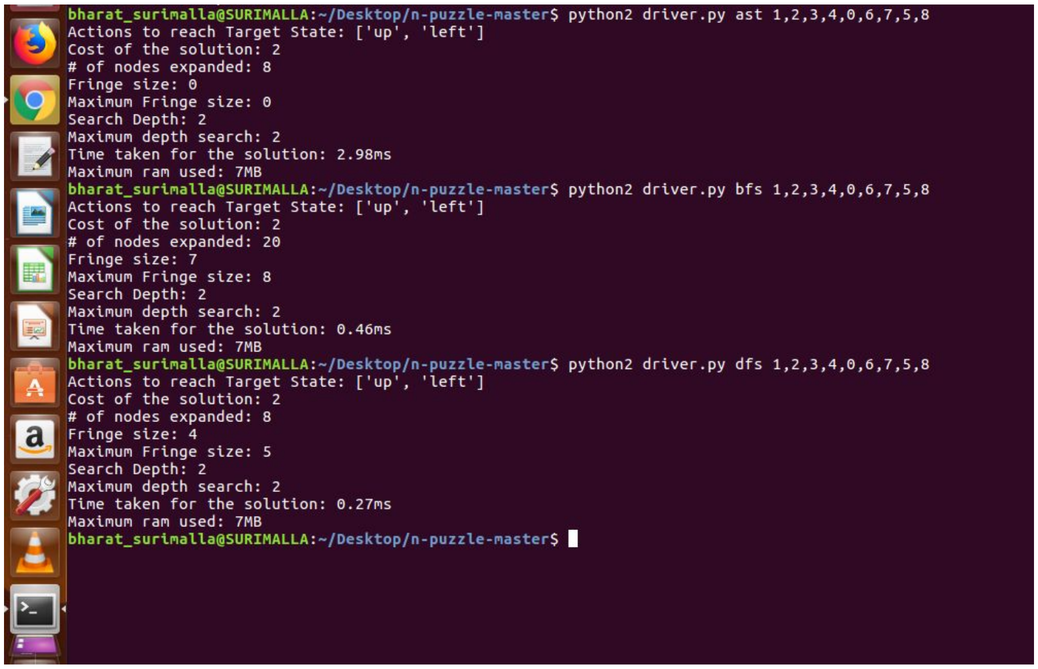


Figure 1. Simple Maze

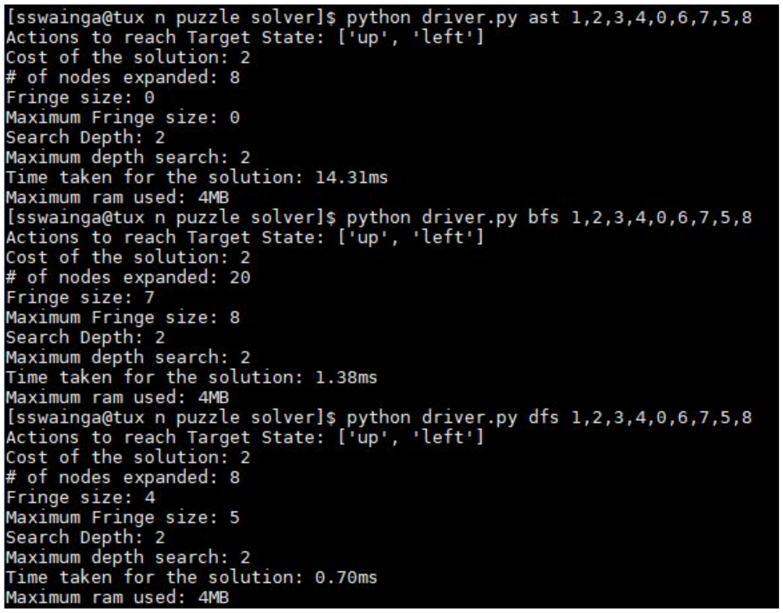
As we can see in figure 1, we started with a simple maze. It needs 2 steps to reach solution. We tried BFS and DFS search algorithm to achieve solution and it was less than 1 ms, whereas for A* search algorithm, it took 2.98 ms (unnoticeable but more compare to other two search techniques).



```
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py ast 1,2,3,4,0,6,7,5,8
Actions to reach Target State: ['up', 'left']
Cost of the solution: 2
# of nodes expanded: 8
Fringe size: 0
Maximum Fringe size: 0
Search Depth: 2
Maximum depth search: 2
Time taken for the solution: 2.98ms
Maximum ram used: 7MB
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py bfs 1,2,3,4,0,6,7,5,8
Actions to reach Target State: ['up', 'left']
Cost of the solution: 2
# of nodes expanded: 20
Fringe size: 7
Maximum Fringe size: 8
Search Depth: 2
Maximum depth search: 2
Time taken for the solution: 0.46ms
Maximum ram used: 7MB
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py dfs 1,2,3,4,0,6,7,5,8
Actions to reach Target State: ['up', 'left']
Cost of the solution: 2
# of nodes expanded: 8
Fringe size: 4
Maximum Fringe size: 5
Search Depth: 2
Maximum depth search: 2
Time taken for the solution: 0.27ms
Maximum ram used: 7MB
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$
```

Figure 2. Result of A*, BFS, DFS on Simple 3x3 Maze

To verify this result, we tried on other remote system and it was same (figure 3).



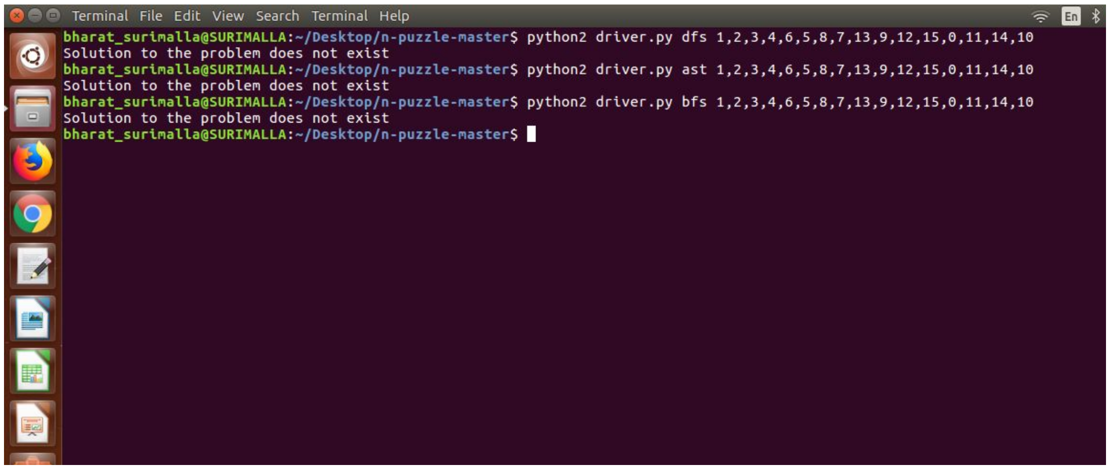
```
[sswainga@tux n puzzle solver]$ python driver.py ast 1,2,3,4,0,6,7,5,8
Actions to reach Target State: ['up', 'left']
Cost of the solution: 2
# of nodes expanded: 8
Fringe size: 0
Maximum Fringe size: 0
Search Depth: 2
Maximum depth search: 2
Time taken for the solution: 14.31ms
Maximum ram used: 4MB
[sswainga@tux n puzzle solver]$ python driver.py bfs 1,2,3,4,0,6,7,5,8
Actions to reach Target State: ['up', 'left']
Cost of the solution: 2
# of nodes expanded: 20
Fringe size: 7
Maximum Fringe size: 8
Search Depth: 2
Maximum depth search: 2
Time taken for the solution: 1.38ms
Maximum ram used: 4MB
[sswainga@tux n puzzle solver]$ python driver.py dfs 1,2,3,4,0,6,7,5,8
Actions to reach Target State: ['up', 'left']
Cost of the solution: 2
# of nodes expanded: 8
Fringe size: 4
Maximum Fringe size: 5
Search Depth: 2
Maximum depth search: 2
Time taken for the solution: 0.70ms
Maximum ram used: 4MB
```

Figure 3. On different system

As we discussed earlier, not all mazes are solvable. So, we tried with some mazes (figure 4) to verify the result and it had no solution (figure 5).

1	2	3	4
6	5	8	7
13	9	12	15
0	11	14	10

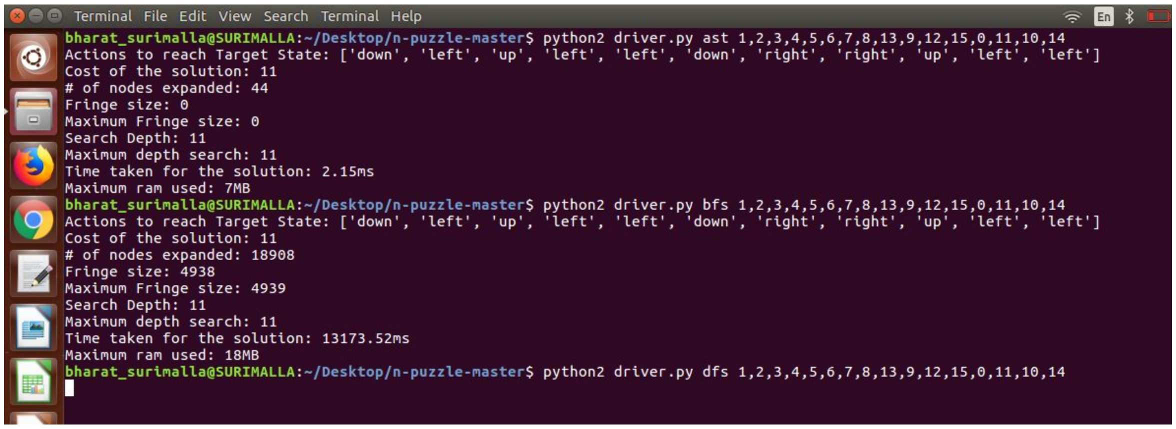
Figure 4. Unsolvable Maze



```
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py dfs 1,2,3,4,6,5,8,7,13,9,12,15,0,11,14,10
Solution to the problem does not exist
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py ast 1,2,3,4,6,5,8,7,13,9,12,15,0,11,14,10
Solution to the problem does not exist
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py bfs 1,2,3,4,6,5,8,7,13,9,12,15,0,11,14,10
Solution to the problem does not exist
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$
```

Figure 5. Result for Unsolvable Maze

We tried to analyze the performance of 3 search techniques with on 4x4 maze and the difference of A* and BFS was quite significant. As we can see in figure 6, A* required around 2 ms to reach solution whereas BFS achieved it after 12 seconds with utilization of 18 mb RAM. Moreover, heuristic algorithm expanded 44 nodes compare to BFS, which expanded around 19,000 nodes, which is quite huge.



```
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py ast 1,2,3,4,5,6,7,8,13,9,12,15,0,11,10,14
Actions to reach Target State: ['down', 'left', 'up', 'left', 'left', 'down', 'right', 'right', 'up', 'left', 'left']
Cost of the solution: 11
# of nodes expanded: 44
Fringe size: 0
Maximum Fringe size: 0
Search Depth: 11
Maximum depth search: 11
Time taken for the solution: 2.15ms
Maximum ram used: 7MB
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py bfs 1,2,3,4,5,6,7,8,13,9,12,15,0,11,10,14
Actions to reach Target State: ['down', 'left', 'up', 'left', 'left', 'down', 'right', 'right', 'up', 'left', 'left']
Cost of the solution: 11
# of nodes expanded: 18908
Fringe size: 4938
Maximum Fringe size: 4939
Search Depth: 11
Maximum depth search: 11
Time taken for the solution: 13173.52ms
Maximum ram used: 18MB
bharat_surimalla@SURIMALLA:~/Desktop/n-puzzle-master$ python2 driver.py dfs 1,2,3,4,5,6,7,8,13,9,12,15,0,11,10,14
```

Figure 6. Result of A* and BFS for 4x4 maze

In most of the cases while implementing 8-puzzle and 15-puzzle with DFS the program is taking more time and memory. There are instances where the program with DFS executed for more than 3 hours and ran out of memory without giving out the final solution. This shows how unfeasible the DFS solution is in few cases. This is mainly because of its searching strategy where it expands deeply on a node till the end. If our system had more processing power, DFS might have given the solution but looking at the usability of the algorithm in such cases it is clearly not a good move to use DFS in such cases.

Conclusion

After implementing various search algorithms to solve n-puzzle we found out few interesting results. Not all algorithms perform well in all situations and not all n-puzzle problems are solvable. BFS and DFS are general search algorithms without any heuristic. They have a fixed solution without any scope for learning whereas A-star algorithm is a heuristic algorithm which learn over time and perform better in finding the solution in most of the cases. The time taken to solve a 8-puzzle game with simple solution is not very different in all the three algorithms whereas the time taken by the 8-puzzle game with complicated solution varies greatly among search algorithms. In a typical 8-puzzle problem with somewhat complicated solution the time taken by A-star algorithm is very less compared to that of BFS and DFS. Also, the memory used the A-stat algorithm to solve the problem is much less than that of BFS and DFS. This is due to learning nature of A-star algorithm and fixed nature of BFS and DFS.The performance of A-star in 8-puzzle game with only one or two moves is slow when compared with DFS and BFS.

In the 15-puzzle game all the algorithms take more time than that of the 8-puzzle game but the results of the 8-puzzle game with respect to algorithms are valid here as well. For a 15-puzzle game with solution including only one or two moves A-star performs slower than that of DFS and BFS and in the cases where the solution is much complex, A-star performs much better that that of BFS and DFS. In some cases DFS algorithm takes quite a long time more than 3 hours whereas A-star gives its output in seconds.

WIth all these results in hand, we can say that algorithms with heuristics perform much better than that of algorithms without heuristics in most of the cases, except few where the solution is very simple and need only one or two steps to reach target state.

References

1. **Identifying optimal N-puzzle solutions using Artificial Intelligence** by *Andrew John Taylor*
2. **Artificial Intelligence - A modern Approach [3rd Edition]** by *Stuart Russell and Peter Norvig*
3. **Breadth-First Search** https://en.wikipedia.org/wiki/Breadth-first_search
4. **Depth-First Search** https://en.wikipedia.org/wiki/Depth-first_search
5. **A* Search Algorithm** https://en.wikipedia.org/wiki/A*_search_algorithm
6. **Solvability of the Tiles Game** by *Mark Ryan*

   http://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html
7. **Artificial Intelligence Tutorial**
   https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_popular_search_algorit
   hms.htm