8 PUZZLE PROBLEM

*

Asis Kumar Rout
Information Technology.)
NITK Surathkal.)
Karnataka,India
asisrout7@gmail.com

Ashok Bhobhiya
Information Technology)
NITK Surathkal.)
Karnataka,India
ashokkumar26042000@gmail.com

Jeeukrishnan Kayshyap
Information Technology.)
NITK Surathkal)
Karnataka,India
geetkayshyap@gmail.com

***Abstract*—The 8-puzzle problem is the largest type of puzzle that can be solved completely. It is simple and yet follows a largely problematic location of 9!/2 states. N-N extension of 8-puzzle is NP-hard. NP-stiffness is non-deterministic polynomial stiffness. In first part of this project, we present the solution of the Breadth first search algorithm. In second part of the project, we are using the 8-puzzle as a model to examine the benefit of IterativeDeepening A*. A glimpse of our results is that all A * implementations is far better than the BFS.**

I. Introduction

An 8 puzzle is a simple game consisting of a 3 x 3 grid (containing 9 squares). One of the squares is empty.The object is to move to squares around into different positions and having the numbers displayed in the "goal state". There exist large variants, e.g. 15 puzzle whatever can be solved, but not to complete. The normal N x N extension of an 8-puzzle is NP-hard.Solution strategies Node ordering schemes in A* search algorithms were motivated by the need for a better understanding of effectiveness. Like any other iteration-intensive search, A * benefits by a good node expansion order, which reduces the time spent in the last (target) iteration. Our results provide evidence that

- BFS takes a lot of time due to the memory limit of normal computer and crashes some time.
- A* algorithm performs better than Breadth First Search.
- Compared to both, A* is better than Breadth First Search to solve this problem.

II. Goals and Objectives

- Main goal of this project is to become familiar with different type of path finding algorithm and analyse them.
- In this project we use two algorithm A* and BFS but A* algorithm is far better than BFS.
- This project is concerned with 8 puzzle sliding problem, in this problem we have to find a goal state from given initial state using minimum cost.
- Get a given initial state and apply different type of algorithm on current state and find the goal state using A* algorithm and BFS algorithm.
- Finally, we compare result that we get from the two different algorithms. Main objective of the project is to determine different type of algorithms and compare them.

III. Literature Survey

[1]. This paper explains the easiest and worst configurations, includes data on the expected solution lengths, and the density and distribution of solution nodes in the search tree. It also presents complete statistical data based on an exhaustive evaluation of all possible tile configurations. 8-Puzzle is used as a workbench model to evaluate the benefit of node ordering schemes in Iterative Deepening A* (IDA*). It was found that sophisticated ordering techniques did not yield better performance, because the 8-puzzle has a low branching factor and it lacks a clear criterion for measuring the goodness of a move. Another result was that almost all IDA* implementations perform worse than would be possible with a simple random ordering of the operators.

[2] In this paper, alternative method of solving n-puzzle problem was proposed. A method for solving n-1 puzzle problem, whereby a leading number drags other set of numbers into their respective positions has been described. The proposed method was used to solve 3, 15, 63 and n-1 puzzles problems. The method was used to derive a generic algorithm, which can be implemented using a suitable programming language. It uses greedy technique by using rotations.

IV. Algorithm

A. A* Search

- The initial state is taken and the next state is found by moving the empty space
- Calculate f-score for each state. (f-score = h-score + g-score)
- Here, h-score is the number of misplaced tiles by comparing the current state and the goal state.
- g-score will remain as the number of nodes traversed from start node to get to the current node.
- This is called expanding the current state.
- After expanding the current state, it is pushed into the closed list(The nodes already visited)
- Newly generated states are pushed into the open list.

- A state with the least f-score is selected and expanded again.
- This process continues until the goal state occurs as the current state.
- The algorithm chooses the best possible action and proceeds in that path.

B. Breadth First Search

- First empty space is moved to its adjacent places.
- In this way new nodes are created .
- Root node is taken as the input in the queue.
- Now, dequeue the root and increase the number of count.
- After ,that children are inserted into the queue.
- We will repeat the fourth and fifth steps till we get the goal state and increase the number of count for every child.
- When the queue becomes empty,count will be the result.We print the total number of nodes visited through the process.

V. Code Overview

- States (state.h: struct State)- In this part we configure the table which have 9 blocks (3*3) And each block has a value between 0 to 8. Since a state can be viewed as a physical representation of the board (game) that's why we include action that is taken for a particular node.
- Operators (state.h: struct Move)- In this part we define the moves like up, down, left and, right. And also define the moveable state i.e. initial and goal.
- Nodes (node.h: struct Node)- We create a node that holds information's about its parent, children, states, path cost, heuristic cost, and depth.
- Linked List Container (list.h: struct NodeList)- It's contained same type of node, as well as pointer to the head and tail nodes.
- Linked List Nodes (list.h: struct ListNode)- This function holds the previous and current node in the list i.e. it linking the two state in list.
- Solution Path (list.h: struct SolutionPath)- This part creates a list of actions to reach the goal node (state), that is, the solution path using singly linked list.

VI. Results and Analysis

The table shows the summary of the performance of BFS and A* under a series of test cases of different difficulty.

TEST CASE	SEARCH ALGORITHM									
	BFS					A*				
	Solution Length	Nodes Expanded	Nodes Generated	Execution time (ms)	Space used (b)	Solution Length	Nodes Expanded	Nodes Generated	Execution time (ms)	Space used (b)
Easy	5	41	77	0	1540	5	5	12	0	240
Medium	9	385	663	0.001	13260	9	17	34	0	680
Hard	12	2251	3783	0.003	75660	12	27	53	0	1060
Worst	-	-	-	-	-	30	4384	7587	0.087	151740

Fig. 1. Comparision between BFS and A* Algorithm

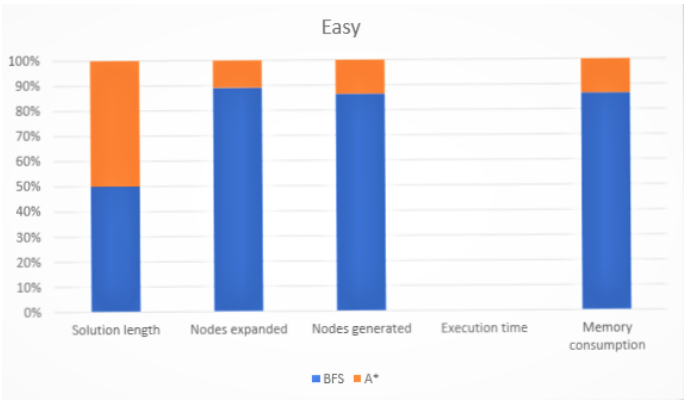

Fig. 2. Easy Test Cases

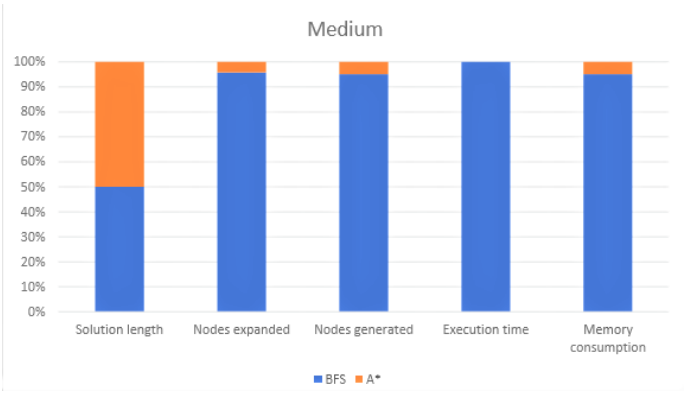

Fig. 3. Medium Test Cases

VII. Acknowledgement

We would like to thank our teachers in our department ,particularly Dr.Bhawana Rudra and J R Shruti for helping us shape the idea of our project.

VIII. Conclusion

- We have implemented two different algorithms on the sliding puzzle problem. The experience we gained from these algorithms is very valuable. Now we will aware of real-life behaviour of these algorithms.
- As we know, informed search algorithms give much better performance on overall solution instead of uninformed search.
- Both the algorithms are able to find solution but the A* algorithm give an optimal solution whereas BFS (Breadth First Search) gives a very insufficient (time taken by BFS is very much as compared to A* algorithm and also number of moves is very high in BFS) solution. Both algorithms are working on the concept of shortest path.
- A* has a list of all the visited node and also a list of that are left to be explored, it finds an optimal node from the list and explore for further, it saves time not to explore unnecessary or less optimal nodes. In BFS algorithm it goes depth by depth and has to find all nodes

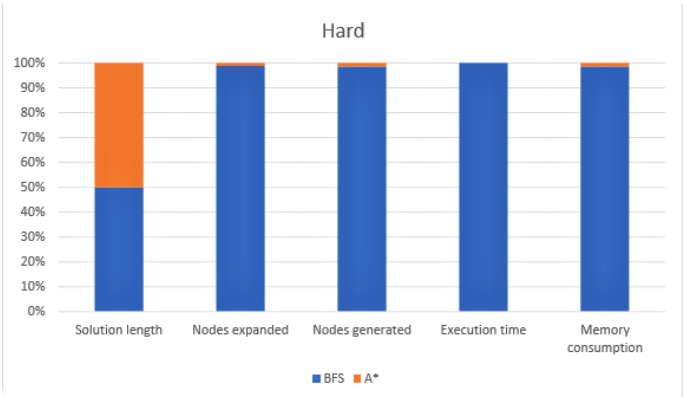

Fig. 4. Hard Test Cases

in a particular depth, so that it takes a lot of time to find shortest path.

- Finally, we can say, yes there are so many more sophisticated algorithms out there but to find a less cost path we can use A* algorithm that is a very sufficient algorithm.
- BFS didn't work for extreme cases due to the memory requirement not being met by our computer.

IX. Screenshots

Fig. 5. 8 puzzle solver

Fig. 6. Input cases

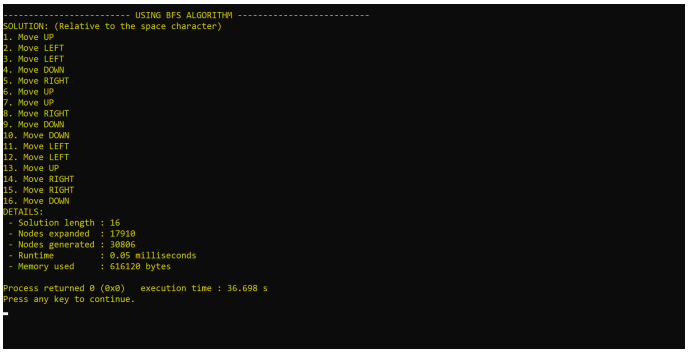

Fig. 7. BFS Algorithm

Fig. 8. A* Search Algorithm

X. References

- [1] Alexander Reinefeld,"Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA"
- [2] OSAGHAE, EO."An Alternative Solution To n-Puzzle Problem"
- [3] P.D.A. Schofield. Complete solution of the 'Eight-Puzzle'.