ANDREA ESPITIA CORREDOR
SID 103165504

Tree Based Search Algorithms for N Puzzle Problem

Abstract

This paper presents an analysis of the solution of N Puzzle problem by implementing tree-based search algorithms: Uninformed and Informed search algorithms, Depth First Search, A* and IDA*. The algorithm techniques and implementation are applied in a given java program base and are described in this report identifying specific cases where an algorithm may be more optimal over another.

Keywords: Search tree algorithms, N puzzle, Informed and Uniformed methods.

1. Introduction

The N-Puzzle problem (sliding puzzle) is a generalisation of the 8-puzzle problem also viewed as a 3x3-puzzle. To solve the N puzzle problem, it is required to move squares tiles arranged randomly in the grid to achieve the desired state goal (in this case organize the numbers in ascending order). The puzzle has one empty square and each tile can slide into the empty space either from top, left, bottom or right.

The N puzzle problem has an initial state of tile arrangements and a goal state to achieve given in an input text file, after each new move a new state is created generating subsequent states that can be represented by a tree graph [1]. The algorithms implemented in the program will traverse the search tree generated when new moves are made in different ways to reach the goal state from the initial state.

This report analyses two Uniformed search algorithm breadth-first search BFS and depth-first search DFS and three Informed search algorithm greedy best-first GBFS, “A Star” A* and Iterative Deeping A* IDA* using given java base code for BFS and GBFS algorithms, adding some improvements on these algorithms and implementing AI search algorithms (Depth first search and A*) to find an efficient solution and analyse/compare their performances in solving the N puzzle problem.

2. Implementation

2.1 Program design

The N-Puzzle problem is designed in a Java code base program using package solver which includes different encapsulated search algorithms in different classes allowing modularization of the program.

The process used by the program is described as follows:

1. NPuzzler class read input file, initiates search methods and generate a result log with number of nodes expanded and path found.
2. An input text file is read with three lines: The first line contains a string NxM: representing the number of rows and the number of columns of the puzzle, the next two lines represent the start-configuration and the end-configuration of the puzzle, respectively. This design enables the different search algorithms to use the same parameters when implementing the search.
3. The initial state and goal state are recorded in PuzzleState class.
4. The algorithm search is applied (by using the method the user wants to use to solve the puzzle calling the implemented Class using the algorithms acronyms BFS, DFS, GBFS, AS, IDA*)
5. If the current state is not the goal state, the program increments node expansion and record the path solution according to the chosen algorithm until a solution is found.
6. If a solution is found the program writes in CLI the number of nodes expanded and the solution path.
7. If a solution is not found the program writes in CLI “No solution found”.

2.2 Algorithm implementations

The 4 search algorithms described in this paper include two Uniformed search algorithms BFS and DFS which do not have the information about the entire state space but only knows how to traverse the tree also known as “blind search”. On the contrary, the two Informed search algorithms GBFS and AS have information of the entire space state obtained by using heuristic estimations allowing more efficient searches. The following section will explain the AI search algorithms techniques used in the software and analyse the results found during its implementation.

Breadth First Search - BFS

Breadth First Search is an uninformed type algorithm. The algorithm records the explored nodes in the frontier queue following the FIFO (First in, First Out) technique and traverses the search tree by expanding all options one level at a time. The result found implementing BFS algorithm in the program using the configuration suggested are described as follows:

Nodes Expanded = 52.497

Solution Path length = 20

The test has expanded 52.497 nodes and found a solution length of 20 moves. The results show that despite BFS can find a short solution length, it has to expand a lot of nodes and record them during the process. The algorithm is optimal as it produces an optimal path however is not the fastest algorithm possible to implement in the N puzzle problem.

Depth First Search – DFS

Depth First Search is the second uniformed type algorithm implemented in the program. DFS explores from the root node and traverse as many nodes as possible on the chosen branch, if the goal has not been found it will backtrack. The data structure used in the DFS defined in the frontier is now a Stack where LIFO (Last in first out) technique is applied. The result found implementing DFS algorithm in the program using the configuration suggested are described as follows:

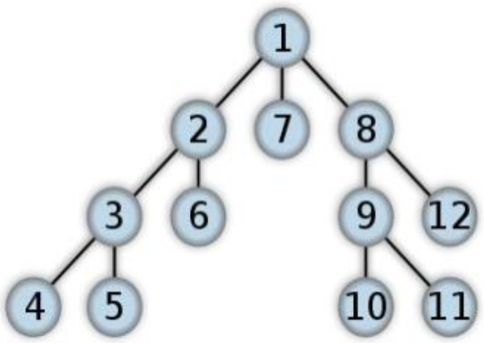


Figure 1. Depth First Search Tree Traversal [4]

Nodes Expanded = 8086

Solution Path length = 7876

The test has expanded 8086 nodes and found a solution length of 7876 moves. The result shows that DFS produces a much longer path to the solution goal however expands less nodes than BFS. DFS is not a complete solution considering during the search process it can explore a branch without getting to an end or reaching the goal state. The algorithm is also not optimal as number moves to reach the solution of the problem is higher.

Greedy best first search - GBFS

Greedy best first search is an informed search algorithm which is based on the Heuristic function to choose the next node to expand. The heuristic function finds the most promising path by estimating the cost of the cheapest path from the root node to the goal node (In this case the number of misplaced tiles were considered as heuristic function of the problem). When traversing the nodes, a list of successors is added to the frontier by comparing the cheapest heuristic value to pick the next node to expand. The result found implementing GBFS algorithm in the program using the configuration suggested are described as follows:

Nodes Expanded = 164

Solution Path length = 36

The test has expanded 164 nodes and found a solution length of 36 moves. GBFS algorithm results represent a lower number of nodes expanded and solution path length considering the implementation of the heuristic function. GBFS algorithm produces the lowest number of expanded nodes than uniformed search algorithms. However, it is not optimized as the cheapest node to explore will go through heuristically suboptimal nodes even when there are other nodes with less path cost to reach the goal state. Furthermore, GBFS is not complete as there is a high chance of the algorithm to get on a lost branch with initial low heuristic values (Similar to DFS algorithm).

A star - A*

A star is an informed search algorithm which implements heuristic-based function and calculates the path cost from the current node state to the new expanded node. A* calculates the heuristic function H(n) = Difference (Goal State, Current State). The algorithm will only expand the most ideal nodes which represents the lower heuristic and path cost to reach the goal.

Nodes Expanded = 522

Solution Path length = 20

The test has expanded 522 nodes and found a solution length of 20 moves. A* algorithm shows that it can find the best solution like BFS algorithm in 20 moves, however it managed to expand a smaller number of nodes. A* is optimal as the heuristic value implemented (Manhattan distance) is admissible as it will never overestimate the cost to reach the goal. It is more efficient and faster than BFS and DFS, however it requires more computation time as it needs to evaluate the heuristic value and the path cost from the root node to reach the goal.

Iterative Deeping A* - IDA*

IDA* is an informed algorithm adapted from A* algorithm to reduce memory consumption. IDA* uses Iterative Deeping algorithm including heuristic function from A* to cut off nodes in the search if the estimate heuristic value of the successor nodes is small and do not exceed the cut off on the previous iteration [1]. While A* keeps track in the queue of unexplored nodes filling up memory space quicker, IDA* traverse the search tree selecting the smallest heuristic value without retaining memory of searched nodes.

Nodes Expanded = 378

Solution Path length = 20

The test has expanded 378 nodes and found a solution length of 20 moves. IDA* algorithm reduces the number of nodes expanded in relation with A* algorithm and provides the same solution path length. The algorithm is optimal in terms of memory considering the reduction of nodes explored. The drawback of IDA* is that due to the absence of memory it may explores nodes again impacting its running time performance.

2.3 Limitations, bugs, optimization features

Known Bugs

– Invalid arguments and incorrect puzzle files will cause the program to crash due to very limited error handling
– Infinite loop created on DFS algorithm

Known limitations:

- Solvability check is not implemented to check whether different tiles arrangement is solvable or not which may improve program performance.
- BFS algorithm is slow. It takes too much time to print algorithm solution.
- The expansion of nodes is not following the technique (When all else is equal the tiles should move the empty cell UP before attempting LEFT, before attempting DOWN, before attempting RIGHT)

Optimizations

- Computing Manhattan Distance instead of computing misplaced tiles. Manhattan distance is an admissible heuristic technique which improves the performance of the implemented informed algorithms.
- State generation check by checking if a state has been visited before, improving performance and saving memory space.

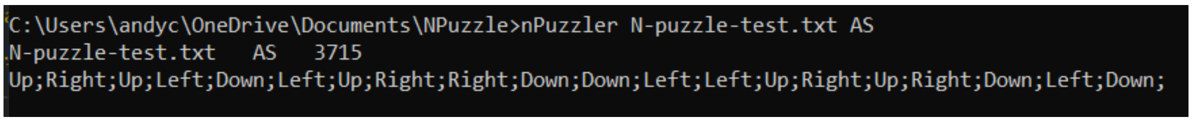
3. Observations

After implementing the 4 search algorithms to solve N puzzle problem the following observations were found:

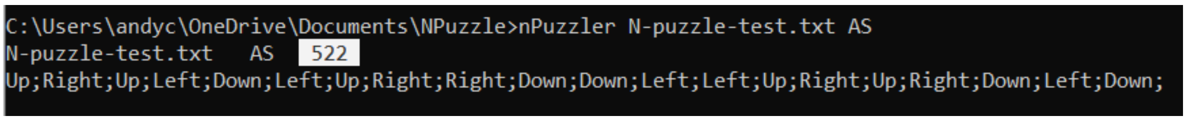
Summary of AI Algorithm Results on the N Puzzle		
Algorithm	**Nodes Expanded**	**Solution Length**
Breadth First Search	52,497	20
Depth First Search	8086	7876
Greedy BFS Search	164	36
A* Search	522	20
IDA* Search	378	20

Table 1. Summary of AI Algorithm Results on the N Puzzle

- Between the two uniformed search algorithms implemented BFS provides the best solution path than DFS even considering the high memory use.
- DFS time taken to find the path solutions is dependent on the right move. If the algorithm moves in a different direction of the suggested one (I.e when all else is equal method), it has to explore all nodes in that branch until the end to then return and explore a new node.
- Repeated state check allows the algorithms to avoid exploring same nodes again, while saving time performance and improving memory management
- Number of nodes iterations visited using A* algorithm is same as BFS algorithm
- A* uses more memory than GBFS algorithm as the expanded nodes were higher.
- A* finds the shortest path with a moderate memory size and overall good time performance
- GBFS uses less memory space than A* but it may not find the most optimal solution in terms of cost
- Implementing Manhattan distance is more admissible than misplaced tiles as every tile will move the number of spots between its position to the goal position.
- The performance of search algorithms depends on the quality of the defined heuristic function. For this problem, when using misplaced heuristic function in A* algorithm the number of expanded nodes is higher as when Manhattan distance is implemented Figure 2.



```
C:\Users\andyc\OneDrive\Documents\NPuzzle>nPuzzler N-puzzle-test.txt AS
N-puzzle-test.txt   AS   3715
Up;Right;Up;Left;Down;Left;Up;Right;Right;Down;Down;Left;Left;Up;Right;Up;Right;Down;Left;Down;
```



```
C:\Users\andyc\OneDrive\Documents\NPuzzle>nPuzzler N-puzzle-test.txt AS
N-puzzle-test.txt   AS   522
Up;Right;Up;Left;Down;Left;Up;Right;Right;Down;Down;Left;Left;Up;Right;Up;Right;Down;Left;Down;
```

Figure 2. A* CLI result using heuristic # misplaced tiles vs Manhattan Distance.

- IDA* algorithm allows having a smaller number of nodes explored than A* as it does not store any explored nodes.
- BFS, A* and IDA* provide same solution path length of 20 moves however memory optimization is different and works better on informed algorithms.

ANDREA ESPITIA CORREDOR
SID 103165504

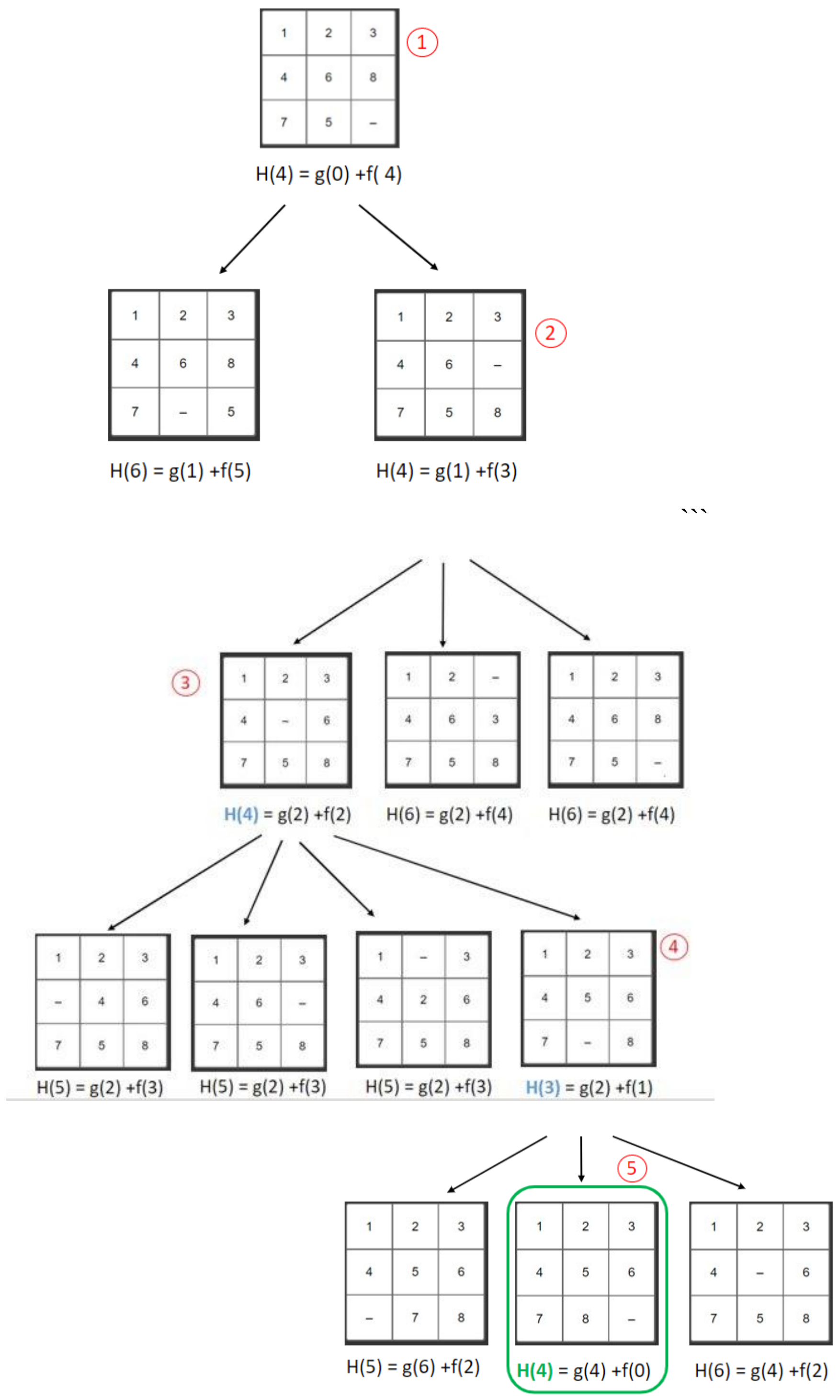


Figure 3. A* traversal search tree example for N Puzzle problem using Manhattan Distance

4. Conclusion and future work

Different tree-based search algorithms were analysed to solve N puzzle problem. Considering the best approaches, A*, BFS and IDA* offers the best solution path length. However, memory efficiency and time performance are different as the most optimal search algorithm to use in this problem requires an informed state space in this case A* algorithm which implements the heuristic function and path cost to provide a more optimal solution. To reduce memory space IDA* can be used to solve N puzzle problem by adapting A* algorithm using deepening depth-first search algorithm with heuristic function.

In complex solutions, A* is the best choice to solve N Puzzle problem in order to avoid possibilities of getting a suboptimal solution from other algorithms that do n ot consider cost when exploring new nodes. A further improvement can be implemented in the program to solve larger size of puzzle NxM puzzle problem.

5. Acknowledgements – Resources

The following resources supported the implementation of the algorithms in the program:

[1] Rusell, S. and Norvig, P. *Artificial Intelligence: A modern Approach Chapters 3 and 4.* 2009. [book] Available at: <https://www.pearsonhighered.com/assets/samplechapter/0/1/3/6/0136042597.pdf> [Accessed 22 March 2022].

Resource: AI a modern approach Chapters 3 and 4 explain techniques used in the program to solve the problem by implementing uniformed and informed tree-based search algorithm. The book includes useful information about 8 Puzzle problem heuristic implementation and describes pseudocode of DFS and A* algorithms which were applied in the program.

[2] Dubreuil, C. and Drogoul, A., 1998. A Distributed Approach To N-Puzzle Solving. [online] ResearchGate. Available at: < https://www.researchgate.net/publication/2710914_A_Distributed_Approach_To_N-Puzzle_Solving> [Accessed 22 March 2022].

Resource: The document includes useful information about N puzzle problem environment helping with decision making of tiles behaviour, how to avoid infinite loops and theorical analysis included in the report

[3] Ai.stanford.edu. 2021. *Heuristic (Informed) Search - Stanford AI Lab*. [online] Available at: < https://ai.stanford.edu/~latombe/cs121/2011/slides/D-heuristic-search.pdf > [Accessed 7 April 2022].

Resource: The document helped me to understand heuristic estimation on N puzzle problem identifying the most admissible option to optimize the search result.

[4] W. Fiset. *Depth First Search Algorithm* | *Graph Theory*. 2018. [video] Available at: < https://www.youtube.com/watch?v=7fujbpJ0LB4 > [Accessed 12 April 2022].

Resource: The video has useful explanation of DFS algorithm including pseudocode details.

[5] Bandyopadhyay, A., 2021. *Implementing DFS in Java* | *Depth First Search Algorithm* | FavTutor. [online] FavTutor. Available at: <https://favtutor.com/blogs/depth-first-search-java > [Accessed 12 April 2022].

Resource: The website explains how the stack data structure works on DFS algorithm including also java code implementation that helped me to understand Linked List functions.

[7] Han, J., 2018. *Solving 8 puzzle with A* search*. [video] Youtube.com. Available at: <https://www.youtube.com/watch?v=GuCzYxHa7iA&t=149s > [Accessed 12 April 2022].

Resource: The video helped me to understand how to implement Manhattan distance, analyse A* algorithm structure and support theorical analysis in the report.

[8] Dang C., 2020. *IDA* search*. [video] Youtube.com. Available at: < https://www.youtube.com/watch?v=Krh2PlkbN28 > [Accessed 15 April 2022].

Resource: The video helped me to understand how IDA* algorithm works, analyse IDA* algorithm structure and support theorical analysis in the report.

[9] Wikipedia.org. 2022. Iterative deepening A* - Wikipedia. [online] Available at: < https://en.wikipedia.org/wiki/Iterative_deepening_A*#Pseudocode> [Accessed 15 April 2022].

Resource: Website helped me to understand pseudocode of IDA* algorithm .