

Single Agent Search Strategies for 15 Puzzle Solver

Deepak Atal
da1722@nyu.edu

Guruprasad
Srinivasamurthy
gs2671@nyu.edu

Shreya Kadambi
sk6407@nyu.edu

ABSTRACT

In this report we attempt to build and analyze the performance of A* and IDA* algorithms to solve the 15 puzzle problem. We propose modifications to the IDA* algorithm to overcome the problem of depth limit. We compare different heuristics by analyzing their performance with IDA*. We define the metrics to assess quality of heuristics by using the number of nodes explored, number of tiles moved, time taken to solve and memory utilized. Through this work we also show the advantages of *linear conflict heuristic* and *on-line learning of costs* as alternate heuristics.

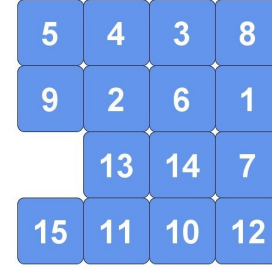
1. INTRODUCTION

A lot of literature on 15 puzzle solver have focused on A* algorithm that builds a single agent search to solve in real time [3][1][2]. Initial state of the puzzle has the tiles scrambled in random order and aim is to slide each tile into the blank location and bring it to the final goal configuration. For this project the final board configuration is set as shown in fig 2. It has been proved in [2] that this is an NP hard puzzle and exactly half of the permutations of tiles can be solved. In our code we also perform a solvability check. There must be even number of transpositions of tiles i.e an even number of tiles whose positions with respect to their neighbor has been swapped. In the rest of the paper we focus on performance of multiple algorithms and heuristics that we implemented for the same. [3] shows that the problem of finding optimum number of moves is NP complete. The time complexity has been shown to scale with the number of tiles as $O(n^3)$.

2. PROBLEM DEFINITION

15 Puzzle Solver can be defined as follows,

1. Initial state :Random positions of tiles . Refer *Figure 1*
2. Action : Swap any neighboring tile with blank space.



A 4x4 grid representing the initial state of a 15-puzzle. The tiles are numbered 1 through 15, with the bottom-right cell (row 4, column 4) being empty. The numbers are: Row 1: 5, 4, 3, 8; Row 2: 9, 2, 6, 1; Row 3: (empty), 13, 14, 7; Row 4: 15, 11, 10, 12.

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

Figure 1: Initial State



A 4x4 grid representing the final state of a 15-puzzle. The tiles are numbered 1 through 15, with the bottom-right cell (row 4, column 4) being empty. The numbers are: Row 1: 1, 2, 3, 4; Row 2: 5, 6, 7, 8; Row 3: 9, 10, 11, 12; Row 4: 13, 14, 15.

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

Figure 2: Final State

If the blank space is in the center there are 4 possible swaps, while the corners can have 3 or 2 swaps. However one of the actions will return to the previous game state. Hence for the center there can be a maximum of 3 swaps.

3. State Space :All possible permutations of tile positions that are solvable*.
4. Goal State :Final tile position to be reached. Refer *Figure 2*
5. Path Cost :Number of times the tiles were swapped with blank position.

2.1 Generation of Puzzles

Initial Game state has been randomly generated. To evaluate and compare the performances of multiple algorithms the solution has been divided in *Easy*, *Medium* and *Hard*

	A*	IDA*
Total Moves	28	28
Time Taken (ms)	85	3010
Memory used(bytes)	33632	1792
Nodes Searched	8408	3606607

Table 1: Comparison of A* and IDA*

based on the minimum number of moves needed to solve the problem. This kind of distinction was necessary as the time complexity varied for each heuristic. Another way of looking at this is, as the game progresses, each game state can be considered as initial state for a new puzzle and complexity evolves from hard to easy. This distinction has been described here as our design parameters take into consideration *hardness* of a puzzle.

2.2 Heuristics

Commonly employed heuristic for 15 puzzle is Manhattan distance which is known to be admissible[1]. Performance of A* was tested for *Manhattan Distance*, it was found to under perform for a certain set of puzzles which involved many moves or complex solution. The heuristic sums the *Manhattan* distance of each tile independently and minimizes the *Manhattan* heuristic for each tile. It does not take into consideration the interdependence of cost of randomizing the neighboring tiles while sliding the tile to the correct position. In this paper we focused on building heuristics that could solve this problem.

3. SEARCH STRATEGIES

3.1 A* Search

The best known algorithm and most visited in literature is A*. It uses the BFS approach to finding optimal solution, however with non uniform costs. The cost for A* algorithm has been defined as

$$COST = g + h$$

. Where, g is the total number of tile moves needed to reach the current board state from initial state. h is the heuristic cost estimated. The cost for each node/game state that was visited earlier needed to be stored. When the total cost on the current branch exceeds the cost on the other visited node, the search started exploring on other states. The needed memory increases as b^N . Time need to solve A* also shows an exponential growth for hard puzzles. The memory constraint pushed the implementation of iterative deepening A* algorithm as presented in the next section

3.2 Iterative deepening A*

The Table 1 below compares A* with Iterative Deepening A*. IDA* used the total costs as threshold instead of a depth limit as in DFS. The algorithm is made to search along each path until the *Goal State* is reached or until the threshold is reached. Refer to fig 3. IDA* opens nodes i, ii, iii and continues in the path until threshold is reached. While A* chooses the most optimal path. As the game progresses IDA* continues on the most optimal path to resemble A*. IDA* consumed less memory. The heuristic estimate of the current state was compared with a threshold to define

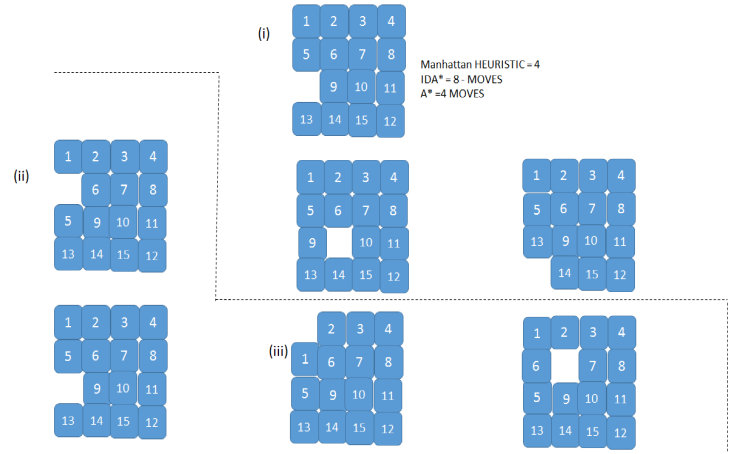


Figure 3: IDA star search tree

the depth limit. This is because it uses the *Depth first* approach. For harder puzzles with more moves IDA* took less memory i.e. linear in nature. IDA* restarts from the root node every time the threshold is increased this causes additional increase in the number of nodes opened. The threshold needed to be carefully designed. It is a trade-off between memory consumption and time.

Threshold: We simulated a modified IDA* where the threshold was varied as the game progressed. Initial search was performed with smaller depths. As the game progressed the threshold was increased to the minimum total cost encountered in the previous run.

Heuristic: Definition of heuristic impacted the choice of threshold value. To identify the game state where the threshold had to be increased the heuristic value was used. The strategy is to define a metric, function of a heuristic that provides us information on game progress.

Pruning: Memory requirements are quite relaxed as we don't need to store any visited nodes. However the tree could be pruned to avoid traversing paths which we are sure would not lead to solution

3.3 Heuristics for IDA*

Heuristics played the most important role in choosing the design parameters. As mentioned, heuristics like Manhattan distance perform well only on easy puzzles. We had to explore better heuristics to combine with Manhattan to reduce the number of nodes visited and time taken to solve.

Hamming Distance: It adds a cost of +1 for every tile whose position has been displaced from its actual goal position. Hamming distance provides a good estimate on the game's progress. However, it performs poor compared to all other heuristics. When calculating the hamming distance the blank tile was not evaluated.

Manhattan Distance: As mentioned before Manhattan is not the best heuristic. Manhattan poorly estimates the cost for different puzzles. For eg, When any two tiles were transposed wrt to each other, the minimum cost needed to swap the tiles back is Manhattan Distance + 2. Manhattan heuristic also generates moves with same costs and this conflict had to be resolved by adding additional conditions.

	NumOfNodes	Time	Moves
Manhattan	375799591	406152 ms	32
Combined Manhattan and Hamming(Threshold 6)	375799591	477900	32
Linear Conflict	32729442	126571 ms	32
Online Learning	12	17	5
Manhattan	18	20	7

Table 2: Heuristics Performance

Combined Manhattan and Hamming Distance: Hamming performs better when the *Game State* is closer to the *Goal State*. See [5]. Hamming was combined with Manhattan as below.

$$Heuristic = Heuristic > threshold ? Manhattan : \max(Manhattan, Hamming)$$

Linear Conflict: An additional cost was added to tiles that were swapped, i.e. when their positions with respect to each other were interchanged. A minimum cost of two was added to *Manhattan* distance. Heuristic estimated is closer to the actual cost. Manhattan assigns the same costs to more than one move in a state and so the algorithm had to resolve the conflict by adding extra conditions to improve the heuristic. Linear Conflict nulls the disadvantages of using Manhattan and Hamming heuristic. When *IDA** uses this heuristic to estimate threshold, it is found to perform much better.

Online Learning of Actual costs: We learn the actual cost to reach the goal in a database for those game states that do not lead to solution while solving a new puzzle. The data base contains patterns that are encountered and their actual costs that were learned. The data base is then queried for the pattern. The actual costs are usually much higher than Manhattan. This performed well compared to Manhattan. Since the Manhattan costs were replaced with actual costs the search along a wrong path terminated upon crossing the threshold much before the pure *IDA**. This reduced the number of nodes searched and the number of moves. For this work, online learning has been implemented to solve the problem where pure *IDA** was sub optimal for threshold values > 10 . Table 2 compares the performance of online learning and Manhattan for *IDA**. As we can see because of DFS nature of *IDA** it takes more moves than optimal for large depth limit values. By replacing each move with actual cost the algorithm reduced the number of moves needed.

```
//Snippet for adding patterns into database
for n over all the neighbor nodes
    IDAStar(node(i) , Threshold)
    If (GoalState())
        break;
    if node(i) is neighbor of root node
        StoreNodeinPatternDatabase(node(i),totalCost)
```

```
//Snippet to get the pattern matching heuristic
if (boardstate matches any pattern in the DB)
    Cost = querythedatabase(pattern) //Higher than Manhattan
else
    Cost = manhattanwithlinearconflict()
```

4. CONCLUSION

We found that *A** algorithm produced quick results for easy puzzles while *IDA** performed better for complex puzzles with larger depths. A modified version of the *IDA** algorithm was also proposed to avoid unreasonable increments of depth limit. For every rerun where the goal state was not reached for a given threshold, the new threshold was updated to the minimum of the total costs encountered in the previous run.

We also found that the linear conflict heuristic gave the best results among all the heuristics for *IDA**. The online pattern matching heuristic provided lesser moves and fewer visited nodes in cases where the optimal moves were not achieved using *IDA**. Pattern matching however needed additional cost of searching the data base. This could be avoided by changing the way patterns are stored and thereby reducing the search space of the data base which has not been shown in this work.

Finding the heuristics was a harder problem as optimal path to each puzzle depended on intermediate states. Features of these intermediate states varied. Finding a common heuristic that caters all the solutions was a tough task. To avoid this problem we propose online learning of features could perform much better.

5. REFERENCES

1. Finding Optimal Solutions to the Twenty-Four Puzzle Richard E. Korf and Larry A. Taylor
2. Efficiently Searching the 15-Puzzle, Joseph C. Culberson and Jonathan Schaeer.
3. Searching with Pattern Databases, Joseph C. Culberson and Jonathan Schaeer. Department of Computing Science, University of Alberta.
4. Real-Time Algorithm for the (n^2-1) Puzzle. Ian Parberry. Department of Computer Sciences, University of North Texas, P.O. Box 13886
5. https://ece.uwaterloo.ca/~dwharder/aads/Algorithms/N_puzzles/