SOLVING SLIDING BLOCK PUZZLE PROBLEM USING ARTIFICIAL INTELLIGENCE CONCEPTS AND ALGORITHMS

Amanta Sunny

Jaydeep Ram

Rishav Chatterjee

Vishwaraj Vala

Course Instructor: Dr. Dan Wu

COMP-8700-2019F

Department of Computer Science

University Of Windsor

ABSTRACT

A sliding block puzzle is a combination puzzle that challenges a player to slide pieces along certain routes (typically horizontally and vertically) on a square board to establish a certain end-configuration. The pieces to be moved may consist of simple shapes, or they may be imprinted with colors, patterns, sections of a larger picture (like a jigsaw puzzle), numbers, or letters. Solving sliding puzzle problems have been attempted by many prominent researchers using mathematical approaches or estimation (heuristic) approaches. In this report, the complex domain of sliding-block puzzle is taken into consideration which offers significant challenges for the field of artificial intelligence. After scrutinizing various properties of this domain, we have shifted the primary focus of our project on Breadth-First Search (BFS), Iterative-Deepening Depth-First Search (IDDFS) and A* search algorithm with heuristic function. A comparative analysis is also provided amongst the above-mentioned algorithms in terms of evaluated time and space complexities for a precise clarification and conclusion.

Keywords: *Sliding Puzzle, Mystic Puzzle, 8 puzzle, 15 puzzle, Informed Search, Uninformed Search, Heuristic Function, BFS, IDDFS, A*, Manhattan Distance.*

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Artificial intelligence (AI) makes it possible for machines to learn from experience, adjust to new inputs and perform human-like tasks. To simply put, it is an area of computer science that emphasizes the creation of intelligent machines that work and react like humans do. Some of the activities computers with artificial intelligence are designed for include, speech recognition, learning, planning and problem solving. Over the years AI programs have managed to achieve complex tasks such as playing a game of chess against human players, playing solitaire and different puzzle problems such as the popular Rubiks cube and sliding puzzle problem. In this project, we discuss in detail our approach towards solving a sliding n-puzzle problem using AI concepts and algorithms such as BFS, A*, IDDFS and further provide comparison amongst the algorithms in terms of time and space complexity.

Chapter 2

LITERATURE REVIEW

1. **"*Notes on the "15" Puzzle*"**,Wm. Woolsey Johnson and William E. Story, American Journal of Mathematics Vol. 2, No. 4 (Dec., 1879).

   In the paper, (Johnson, 1879), the author tackled a specific configuration of a fifteen-puzzle problem (where all the pieces were serially arranged, except 14, 15 had interchanged positions) using mathematical approach which is completely independent on the position of the blank piece. To be able to solve such a configuration constraints and rules were added to the problem by the author. The solution of the entire problem was based on natural and reverse natural arrangements of the board. Other prominent authors such as (Spitznagel, 1967) and (Wilson, 1974) also proposed their individual approaches to solve the problem.

2. **"*Finding Optimal Solution to the twenty four- puzzle*"**, Richard. E. Korf and Larry A, Taylor, computer science department, university of California LosAngels.

   In the paper, (Korf, 1985), the author presented a heuristic iterative-deepening depth-first algorithm that is capable of finding optimal solutions to randomly generate instances of the 15 puzzles, within practical resource limits. The algorithm has also been used successfully in chess programs. The author in (Korf, 1985) further proves that iterative-deepening depth-first is asymptotically optimal among brute-force tree searches in terms of time, space, and length of solution.

Chapter 3

PROBLEM STATEMENT

The n-puzzle sliding problem is a classical problem involving algorithms and heuristics. For the initial state, we have a square board of n number of sliding pieces and a blank space. The n pieces are numbered from 1 to n and are arranged randomly.

Figure 3.1: Initial (left) and Goal (right) state of a 15-puzzle problem.

The individual pieces can only be moved horizontally and vertically. The goal of this problem is to arrange all the pieces in order from top to bottom and left to right, so that the result contains a square board with all the pieces arranged serially and in ascending order. Figure 1 illustrates the initial and goal state of a 15-puzzle problem.Given an initial state of the board, the search problem is to find a sequence of moves that transitions this state to the goal state. The search space is the set of all possible states reachable from the initial state. A state is an instance having a specific arrangement of the board. There can be multiple states from initial to goal states depending on the approach used.

In this report we focus on BFS, IDDFS and A* algorithm with heuristic function and draws together the above two review approaches by describing how heuristic information from the problem domain of finding the minimum cost path through a graph, can be incorporated into a formal mathematical theory of graph searching and demonstrates an optimality property of a class of search strategies. It also presents a general algorithm which prescribes how to use such information to find a minimum cost path through a graph.

Chapter 4

PROPOSED APPROACH

The proposed system we discuss our approach towards solving a n-puzzle sliding problem. We have implemented different search algorithms such as Breadth-First Search, A* search with Manhattan Distance Heuristic and Recursive and Non-recursive Iterative Deepening Depth-First Search to find the most optimal solution. We briefly explain the algorithms used and the steps involved in the process.

4.1 Breadth-First Search (BFS)

Breadth-first search is a traversing algorithm (i.e. visits every vertex and edge exactly once) that traverses a tree like structure in the order of its breadth i.e. in a layer-wise fashion, exploring all neighboring nodes in one layer before moving to the next layer. BFS algorithm tends to find the shortest path in a graph. For the given n puzzle problem, BFS algorithm would start at an initial point known as the root node of a tree and traverse layer by layer to all its children nodes. Figure 4.1 illustrates last stage traversing a BFS algorithm being implemented on a tree structure The following step are involved in shuffling and applying

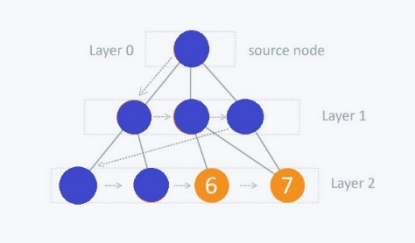

Figure 4.1: BFS traversing the last layer of the tree

the BFS algorithm to solve the n-puzzle problem:

- We remove a node from the frontier set.
- We check the state against the goal state to determine if a solution has been found.
- If the result of the check is negative, we then expand the node. To expand a given node, we generate successor nodes adjacent to the current node, and add them to the frontier set. Also, if the successor nodes are already in the frontier, or have already been visited, then they should not be added to the frontier again.

A* Search with Manhattan Distance Heuristic

A* search algorithm falls into the category of informed search that tend to be highly efficient as it consumes less time to find a solution and the shortest distance possible. However, such algorithm makes use of a heuristic function to calculate the estimated distance. A heuristic is a function that estimates how close a state is to the goal state and helps to determine a solution in reasonable time by intelligent expansion of nodes. For our n-puzzle problem we will be using A* along with Manhattan Distance Heuristic. Manhattan Distance Heuristic helps the search algorithm to estimate the distance, by calculating the sum of vertical and horizontal distance to be traveled, to reach the goal state from current state. Figure 4.2 represents the implementation of A* search with a heuristic function on a graph having 6 nodes, a being the starting node and z being the goal state.

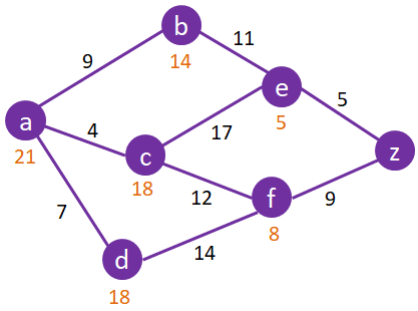

Figure 4.2: Representation of A* search with heuristic function on a graph.

In Figure 4.2, the numbers colored in orange are the estimated heuristic function values that gets smaller as the algorithm approaches near the goal state(z). Numbers marked in black are the cost to reach the goal state(z), and the overall cost increases, the farther the current node is from the goal state. A* search uses a sum of both the values to find the most optimal solution to the problem.

4.2 Iterative Deepening Depth - First Search (IDDFS)

IDDFS combines the characteristics of both depth-first search (DFS) and breadth-first search (BFS). A combination of such leads to better results as we can get the memory efficiency of depth-first search and faster search process of breadth-first search. While running the IDDFS algorithm, we call DFS algorithm for identifying different depths starting from the initial state. It will traverse till the depth mentioned and try to find the desired state. If the desired state is not found, then it would increase the depth index by one run recursively. We can observe that with this approach, the first node will be visited multiple times whereas the ones on the last level will be visited once and second to last level will be visited twice and so on. Figure 4.3 is a representation of IDDFS algorithm. In Figure 4.3, we can see a tree graph with 3 layers and the depth

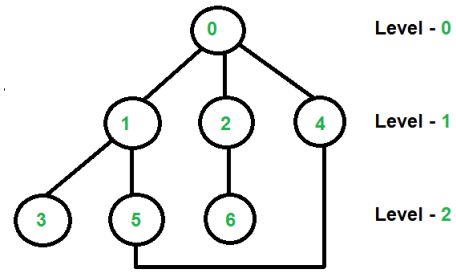

Figure 4.3: Representation of IDDFS at depth limited to 3.

of IDDFS is limited to 3. Hence the search result for the first layer would be 0, for second layer it would be 0,1,2,4 and finally for the last layer it would be 00,1,3,5,2,6,4,5. There is also a recursive call on the above graph so there will be another set of search results such as 0,1,3,5,4,2,6,4,5,1.

As we can see that IDDFS expands all nodes at a given depth before expanding any nodes at a greater depth, thus it is guaranteed to find a shortest-length solution. Also, since at any given time it is performing a depth-first search, and never searches deeper than depth d, the space it uses is O(d). The disadvantage of IDDFS is that it performs wasted computation prior to reaching the goal depth.

Chapter 5

EXPERIMENTAL SETUP

First, we tested the BFS algorithm on an 8-puzzle sliding problem. On an average it took just over a minute to find a solution to the problem. The best possible solution had a cost of 14 and traversed 2081141 nodes. However, the output depends on the complexity of the arrangement of the board (Figure 5.1). For

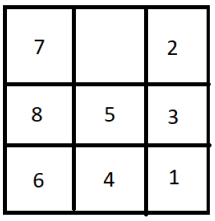

Figure 5.1: Unsolvable 8-puzzle Configuration

increased size of the puzzle where n = 15, the result started becoming the worse. The compilation took longer, and the memory started getting consumed by process threads, leading to an indefinite wait-time, which seemed to be an in-efficient and non-optimal approach.

When trying the implementation of IDDFS on 8-puzzle problem, the results were similar to the BFS approach, however slightly faster but ended up consuming a bit more memory than BFS. While switching to 15-puzzle problem, the results seemed even worse than 8-puzzle. The program would execute for an indefinite period and at a point the execution had to be aborted due to entire consumption of computer memory. However,based on complexity of arrangement of the board the results can vary.

A* search using Manhattan Distance performed the best out of the other approaches for both 8-puzzle and 15-puzzle combinations. The execution time for 15-puzzle was in under a minute and memory consumption was in KB (kilobytes). On 15-puzzle combinations, nearly 8000 positions were examined and

the path cost for the best result was 52 using A* with Manhattan Distance. The procedure was based on the following formula,

* **f(n)=g(n)+h(n), where, f(n)** = total estimated cost of path through node n, **g(n)** = cost so far to reach node n **h(n)** = Heuristic function (Manhattan Distance)
* **h(n) =** $|x1 - x2| + |y1 - y2|$**, where, x1** is coordinate of goal node, **x2** is coordinate of current node along x axis, **y1** is coordinate of goal node and **y2** is coordinate of current node along y axis

5.1 Discussion And Comparison

To summarize, A* with the Manhattan Distance Heuristic produced the best results. If a board with complex arrangement is passed in, BFS and IDDFS will not work. This also holds true for higher values of n such as n 8. The runtime will timeout after a few seconds due to high memory consumption. A* search using Manhattan Heuristic program used for this project can complete 99 percentage of all board states that is excluding an unsolvable board configuration. However, for BFS and DFS, breaking the problem into sub-problems may help to produce slightly better output for n having higher value, but at the cost of efficiency and simplicity. The following table illustrates the results of all the approaches for 8 puzzle for BFS and IDDFS,15 puzzle for A* and images show the output either 15 or 8 puzzle problem(because few algorithm eats up a lot of memory space for 15 puzzle):

```
In [11]: runfile('C:/Users/HP/Desktop/ai_project/bfs_.py', wdir='C:/Users/HP/
ai_project')



************************************************
PUZZLING COMPLETE
************************************************


[[1, 2, 3], [4, 5, 6], [7, 8, 0]]

Elapsed Time:  76.484375

Nodes traversed: 2081141

Total Cost: 14

In [12]:
```

Figure 5.2: BFS for 8 puzzle

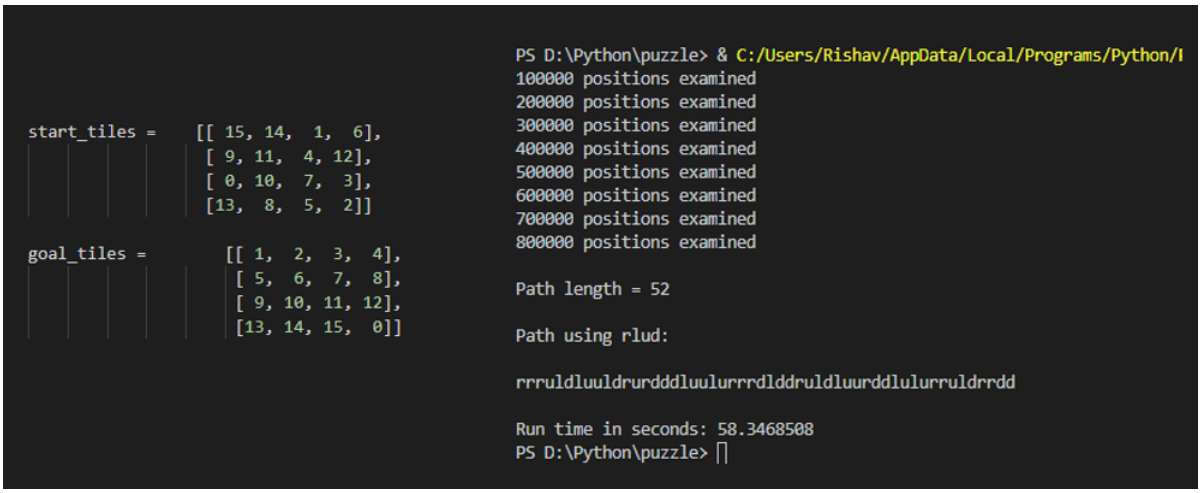

Figure 5.3: A* for 15 puzzle

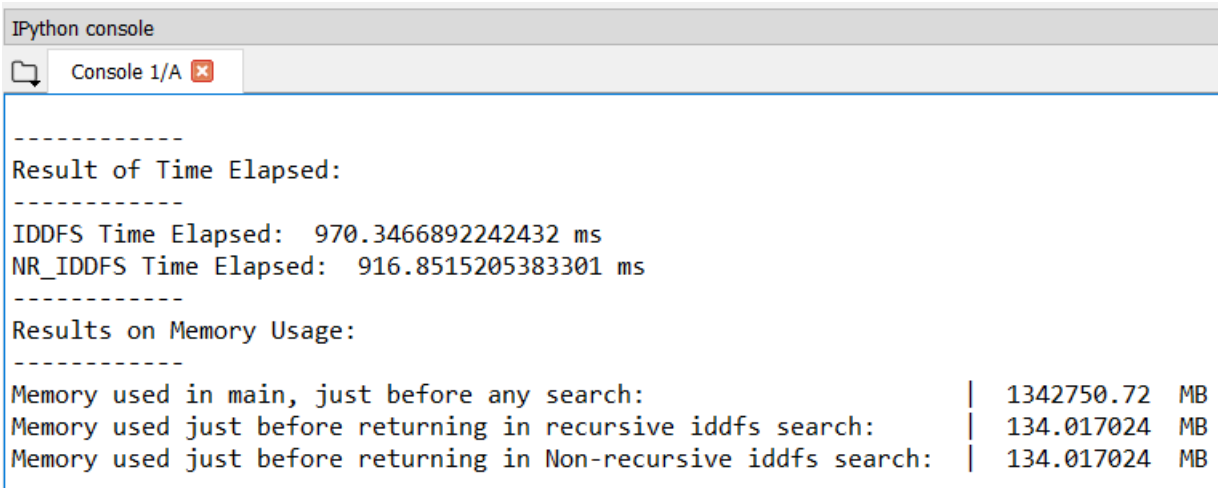

Figure 5.4: IDDFS for 15 puzzle

Algorithms Used	**Board Configuration(n)**	**Time Consumed(avg)**	**Memory Consumed(avg))**
Breadth-First Search	8	76.484 sec	143 mb
A* search using Manhattan Distance	15	58.346 sec	<< 1mb
IDDFS (Recursive)	8	72.34 sec	145.46 mb
IDDFS (Non-Recursive)	8	72.11 sec	145.20 mb

Table 5.1: Comparison Of Algorithm

Chapter 6

CONCLUSION AND FUTURE WORKS

To conclude, in this project to solve a n-puzzle sliding block problem using multiple approaches we came across a few design considerations such as BFS and IDDFS are not optimal to be used for solving puzzle problems having high n values. We also encountered technical backdrops such as BFS and IDDFS running out of memory during its execution to solve a 15-puzzle problem and running for an indefinite period until the execution had to be aborted.

Thus, it is to be noted that BFS and IDDFS both belonging to the category of un-informed search performed significantly worse than A* search algorithm which is an informed search method as it uses the application of heuristic functions. In our case, we used Manhattan Distance heuristic which performed quite well, however few other heuristic functions exists which may provide even better solution such as Euclidean Distance and Diagonal Distance. Since, A* performed well in our cases for 8 and 15-puzzle, it would be interesting to see how well it performs for even higher values of n such as n = 24, 35. Application of other AI concepts such as machine learning can be used which may provide even better path finding solutions is also yet to be explored.

REFERENCES

- Archer, A. F. (1999). A Modern Treatment of the 15 Puzzle. American Mathematical Monthly, Vol.106, pp. 793-799.
- Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, Vol. SSC4(2), pp. 100107.
- Johnson, W. W. (1879). Notes on the '15 Puzzle I. American Journal of Mathematics, Vol. 2, pp. 397-399.
- Johnson, W.W., and Storey, W.E. (1879). Notes on the 15 Puzzle. American Journal of Mathematics, Vol. 2, pp. 397404.
- Korf, R.E. (1985). Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence, Vol. 27(1), pp. 97109.
- Korf, R.E., and Taylor, L.A. (1996). Finding Optimal Solutions to the Twenty-Four Puzzle. In the Proceedings of the 13th National Conference of AI, AAAI, Vol. 2, pp. 12021207.
- Liebeck, H. (1971). Some Generalizations of the 14-15 Puzzle. Mathematics Magazine, Vol. 44, pp. 185-189.
- Loyd, S. (1959). Mathematical Puzzles of Sam Loyd, Vol. 1, pp. 19-20,
- Ratner, D. and Warmuth, M. (1990). Finding a Shortest Solution for the -Extension of the 15-Puzzle Is Intractable. Journal of Symbolic Computation, Vol. 10, pp. 111-137.
- Spitznagel, E. L. Jr. (1967). A New Look at the Fifteen Puzzle. Mathematics Magazine, Vol. 40, pp. 171-174.
- Story, W. E. (1879). Notes on the '15 Puzzle. II. American Journal of Mathematics, Vol. 2, pp. 399-404.

Appendix A

WORK DISTRIBUTION

The project of Solving Sliding Block Puzzle Problem Using Artificial Intelligence Concepts Algorithms was distributed among the members of **Team AImbots** as follows:

Amanta Sunny : working and implementation of the n puzzle using the BFS algorithm and the report on the BFS algorithm along with conclusion and future works.

Jaydeep Ram : working and implementation of the n puzzle using the IDDFS non recursive algorithm and the report on the algorithm along with introduction.

Rishav Chatterjee : working and implementation of the n puzzle using the A* algorithm with Manhattan Distance Heuristic and the report on the algorithm along with Literature review.

Vishwaraj Vala : working and implementation of the n puzzle using the IDDFS recursive algorithm and the report on the algorithm along with abstract.