N-Puzzle with AI

Hoang Do  
University of Massachusetts LowellLowell, MA  
Hoang\_Do@student.uml.edu

William Awad  
University of Massachusetts LowellLowell, MA  
William\_Awad@student.uml.edu

*Abstract*—This document demonstrates the application we designed for the n-puzzle game, in which we can use the algorithms A\* and IDA\* with two heuristics to solve the puzzle. We can also use the application to test the heuristics and compare them with a detail report.

Keywords—A\*, IDA\*, heuristic, GUI

# Introduction

One of the most basic problems since the beginning of artificial intelligence has been the N-puzzle. The simplest form of the puzzle, also known as the 8-puzzle, consists of an area divided into 3x3 grid containing 8 numbered tiles with one empty grid. With the initial state being given, the goal of the puzzle is to reach its goal state. In order to accomplish this task we have used two informed search methods which are the A\* algorithm and IDA\* algorithm. There are two heuristics involved that were used in conjunction with the algorithms to search for the goal state.

# LITERATURE REVIEW

## N-Puzzle

The puzzle was invented by Noyes Palmer Chapman, a postmaster in Canastota, New York, in 1874. This precursor version of the 15-puzzle consisted of 16 tiles in a set square with each row and each column adding up to 34. Chapmans son, Frank Chapman, later brought copies of the enhanced Fifteen Puzzle to Syracuse, New York. From there, by word of mouth as well as growing popularity, the 15-puzzle made its way to Watch Hill, Rhode Island. As the popularity continued to grow, the puzzle made its way across New England where it was picked up by The American School for the Deaf in Hartford Connecticut which began manufacturing the puzzle. [3]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Observable** | **Deterministic** | **Episodic** | **Static** | **Discrete** | **Agent** |
| Fully | Deterministic | Episodic | Static | Discrete | Single |

Table 1. Task environment of N-Puzzle

N-Puzzle is fully observable because the agent can access to all information in the environment relevant to its tasks. It is also a deterministic environment because the outcome can be determined base on the current state. It is episodic because there is only one goal for this problem, and we will try to reach that goal. The environment is not changing and always observable making it static. It is a discrete single agent because the environment is a fixed location with only one changing variable (empty space).

The puzzle game has the initial state and the goal state as the picture below. The state space is depending on the number of n: n2!. For example: if n = 3 (8-puzzle), the state space is 9! = 362880; if n = 4 (15-puzzle), the state space is 16! = 20922789888000. But only have of these states are reachable from any given states. Hence if n =3, there is 9!/2 solvable problems instance. [4]

## A\*

The A\* algorithm is a best-first search algorithm in which the cost linked to a state is f(n) = g(n) + h(n). where g(n) is the cost of the path traversed from the initial state to node n. h(n) is the estimated path-cost or the heuristic function cost from node n to the goal node.

The primary use of the A\*-algorithm is for path searching as well as graph traversal. By using uniform-cost search and heuristic based search, the optimal traversable path between states (nodes) is plotted. [3]

A\* Algorithm: [3]

1. while OPEN contains a node do{

2. delete from OPEN the node <j,p> with lowest f value.

3. place <j,p> on CLOSED.

4. if „j‟ is goal state ,return goal( path p).

5. for each edge e connecting „j‟ and „k‟ with cost e do{

6. if < k,q> is in closed and {p|e } is cheaper than q

7. delete „k‟ from CLOSED.

8. put <k,p|e > in OPEN. End of if.

9. else if <k,q > is in OPEN and {p|e} is cheaper than q

10. replace <k,p|e > on OPEN. End of else if

11. else if k is not on OPEN

12. put <k,p|e> on OPEN. End of else if. End of for

13. end of while

14 .return failure

15. end of algorithm

## IDA\*

In IDA \* a series of depth-first searches with successive increased cost-bounds are performed in order to find the optimal path of traversal. Similar to A\*, the total cost f(n) of a node n is f(n) = g(n) + h(n), in which the g(n) is the cost of reaching that node plus the estimated cost of the path to a goal state h(n). The initial cost bound is set to the heuristic estimate of the initial state, h(root). The bound is increased for each iteration to the minimum path value that exceeded the previous bound. IDA \* is guaranteed to find a shortest solution path if there is an admissible heuristic h. [1]

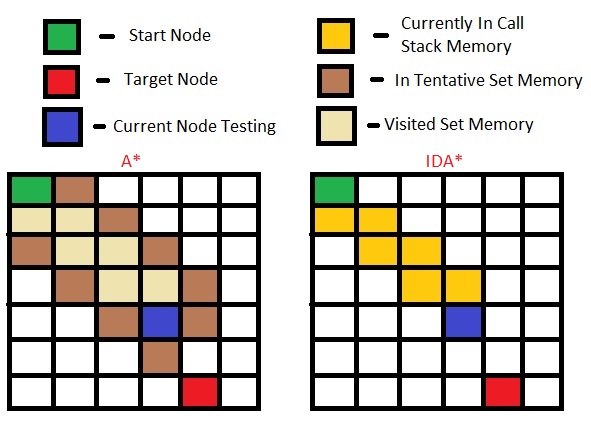
IDA\* Algorithm: [3]

1.start with the f(limit)=f(start)

2.Prune any node if f(node)>f(limit). Do a DFS.

3.Next f-limit=minimum of any node pruned. Go to step 1 and repeat the whole process until the goal node is found.

## Comparation A\* and IDA\* [2]



The A\* search algorithm is less efficient to IDA\* as it keeps a large queue of unexplored nodes. When applying this to the N-puzzle problem, this large allocation of memory quickly lowers the efficiency of the programs ability to reach the goal state. IDA\* makes up for the lack of A\*’s low efficiency by only storing the nodes on its current path requiring the memory that is only linear to the length of the solution. [2]

# METHODOLOGY

Trivial search algorithms (Depth-First-Search, Breadth-First-Search, etc.) require the program to traverse through its path until eventually finding the goal state. To improve upon the search algorithms efficiency, heuristic algorithms are implemented. Heuristic algorithms compute the most efficient cost to reach a specific goal state from any other state.

## Heuristic 1 [1]

Heuristic 1 - Manhattan distance: is the total shortest distance to move a wrong position tile to its right position.

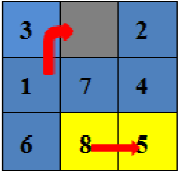


Figure 3.1: move 1 and 8 to their right position

.

rw, cw are row and column in wrong position.

xd = |cw – cr|

yd = |rw – rr|

d = xd + yd is the shortest path to move a tile to its right position

* h1 = d

In the figure 3.1, to move the tile has number 1 to its right position: d = |cw – cr| + |rw – rr| = |0 – 1| + |1 – 0| = 2

Using the same method to others tiles we have:

h1 = 1 + 0 + 2 + 1 + 1 + 0 + 1 + 1 = 7

## Heuristic 2 [5]

Heuristic 2: is the total shortest distance to move a wrong position tile to its right position (same as heuristic 1) plus the penalty value for each pair of tiles in wrong positions.

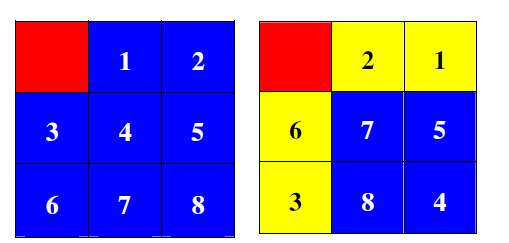


Figure 3.2: two pairs of tiles in the wrong position (2 and 1) and (6 and 3).

* h2 = d + a

Assume: a is the penalty value for each pair of tiles in wrong positions. For each pair, a = 2.

In the right puzzle in figure 3.2, there are two pairs of tiles in the wrong position, so a = 4

## Example of two heuristic

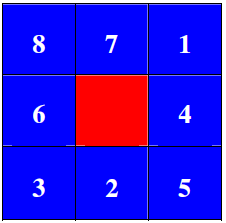


Figure 3.3: One state of the puzzle

From the state of the puzzle above:

+h1 = 4 + 2 + 1 + 1+ 1 + 1 + 3 + 1 = 14

+h2 = h1 + 2 = 16 (one pair wrong position 6 and 3)

# RESULTS

## GUI

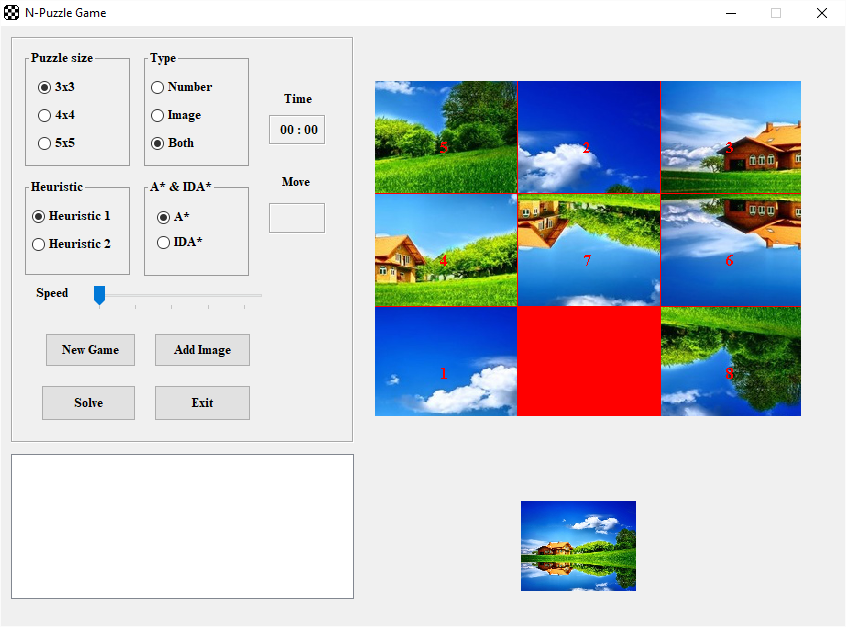


Figure 4.1: GUI for n-puzzle

We have developed the GUI for the application where the user can play by move the red tile to complete the puzzle when it fits the suggestion small image or the goal state (number).

Player can choose the level of difficult by changing the puzzle tile to 3x3, 4x4 or 5x5 corresponding to easy, normal or hard. Player also can use the algorithms A\* or IDA\* and use the heuristic to solve the problem by click the “Solve” button. When using the solve button, the application also provides times, steps and node expanded in the report sections.

## 8-puzzle (3x3)

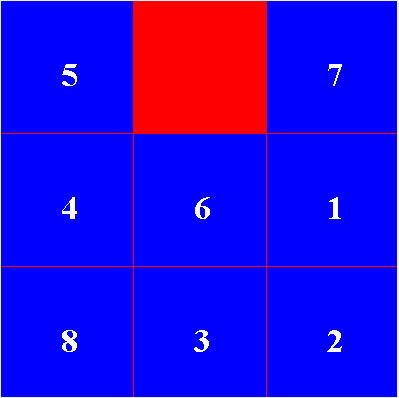


Figure 4.2 Example of 3x3 problem

#### A\*

+ Heuristic 1:

Nodes already evaluated: 1021

Nodes in a tree: 1701

Moves: 23

Time: 12ms

+ Heuristic 2:

Nodes already evaluated: 848

Nodes in a tree: 1428

Move: 23

Time: 10ms

#### IDA\*

+ Heuristic 1:

Nodes already evaluated: 435

Nodes in a tree: 719

Moves: 23

Time: 2ms

+ Heuristic 2:

Nodes already evaluated: 367

Nodes in a tree: 612

Move: 23

Time: 1ms

## 15-puzzle (4x4)

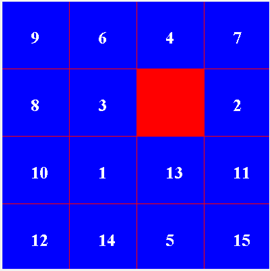


Figure 4.3 Example of 4x4 problem

#### A\*

+ Heuristic 1:

Nodes already evaluated: 10401

Nodes in a tree: 22179

Moves: 31

Time: 524ms

+ Heuristic 2:

Nodes already evaluated: 9010

Nodes in a tree: 19287

Move: 31

Time: 387ms

#### IDA\*

+ Heuristic 1:

Nodes already evaluated: 5907

Nodes in a tree: 12523

Moves: 31

Time: 10ms

+ Heuristic 2:

Nodes already evaluated: 5182

Nodes in a tree: 10984

Move: 31

Time: 10ms

## 25-puzzle (5x5)

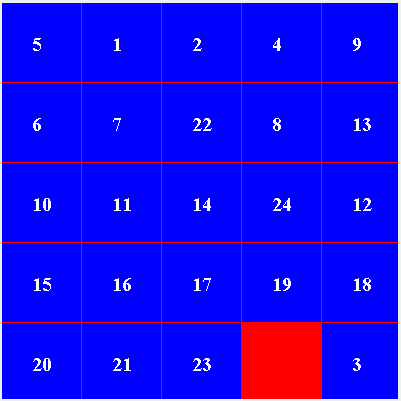


Figure 4.3 Example of 4x4 problem

#### A\*

+ Heuristic 1:

Nodes already evaluated: 15933

Nodes in a tree: 35283

Moves: 35

Time: 1663ms

+ Heuristic 2:

Nodes already evaluated: 12273

Nodes in a tree: 27183

Move: 35

Time: 968ms

#### IDA\*

+ Heuristic 1:

Nodes already evaluated: 15587

Nodes in a tree: 34501

Moves: 35

Time: 78ms

+ Heuristic 2:

Nodes already evaluated: 11863

Nodes in a tree: 26278

Move: 35

Time: 38ms

# DISCUSSIONS

As heuristic2 = heuristic1 + a, so h1(n) h2(n) h\*(n). Heuristic 2 would have less the nodes expanded (Nodes already evaluated) and less time consuming to solve the problem than heuristic 1. So h2(n) is more efficient than h1(n).

The optimization of algorithm A\* depends on the heuristic h(n), and it is only suitable for small state space. If the state space is finite and there is a solution to avoid the repeating checking the states again, then the algorithm A\* is completed (can find the solution), but not guarantee the optimization. If the state space is infinite or there is no solution for avoid checking the states again, then A\* is incomplete (cannot find the solution).

IDA is more efficient than A in terms of memory and time. In general, IDA is faster than A, but it also does not guarantee optimization.

# CONCLUSION

Through understanding and researching this topic, we have a more comprehensive view of the application of artificial intelligence to solve real problems. Because of limited research time, we can only compare A\* and IDA\* together with two heuristics. In the future, we can use this application to test and compare many different algorithms as well as many other heuristics to find the best solution for N-puzzle.

##### References

1. Alexander Reinefeld. “Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA\*.” Paderborn Center for Parallel Computing Waxburger Str. 100, D-33095 Paderborn, Germany, 1993
2. Culberson, Joseph C, and Jonathan Schaeffer. “Efficiently Searching the 15-Puzzle.” The University of Alberta, May 1994. Technical Report TR 94,08 May 1994
3. Desbasish Nayak. “Analysis and Implement of Admissible Heuristic in 8 Puzzle Problem.” Department of Computer Science and Engineering National Institute of Technology Rourkela, India, May 2014
4. Johnson, Storey. “ Notes on the 15'-Puzzle”. Amer. J. Math. 2(1879), 397-404, 1987.
5. Korf, Richard E, and Larry A Taylor. “Finding Optimal Solutions to the Twenty-Four Puzzle.” Computer Science Department, University of California, Los Angeles, 1996.