2/12/2019

Project 1:

The 8 – Puzzle Solver

Harsha Paruchuri CS 205: Artificial Intelligence

SID: 862122817 Instructor: Dr. Eamonn Keogh

Mail: [hparu001@ucr.edu](mailto:hparu001@ucr.edu)

# **Introduction**

This Java application to solve the 8-puzzle was created for the first project in the CS205 Artificial Intelligence Course by Prof. Eamonn Keogh at the University of California Riverside.

The application solves the 8 - puzzle. and provides a choice of three algorithms for the same. The user can select between the Uniform Cost Search, A\* using Manhattan distance heuristic and A\* using misplaced tile heuristic.

The application was developed using Java (version 7) and has an interactive interface that works through the standard output, the input from the user is taken through the standard input.

# **Algorithms**

The initial state is first expanded in all its possible subordinate states by checking if the blank space can be moved left, right, up and down. Out of all the valid moves new states are formed. Of these new states, the best state to expand is chosen using the cost calculated by the algorithm chosen by the user. In each iteration, the state is checked for a match with the goal state and for repeated states.

## **Uniform Cost Search**

This method searches the branches with the same cost. It is basically the A\* algorithm with h(n) set to zero. The cost of expanded node is g(n) which is one.

## **Misplaced Tile Heuristic**

In this method, each non-blank element in the current state is compared to the goal state and if, it is different, the ‘h’ value is incremented. After traversing through the entire grid, the final ‘h’ value is set to that node. For example,

**Goal State: 1 2 3 Puzzle State: 1 2 0**

**4 5 6 4 5 3**

**7 8 0 7 8 6**

The underlined elements are the non-matching ones between puzzle state and goal state. The blank element in the puzzle state is ignored. Therefore, the misplaced tile heuristic is **2**. The h(n) value for this puzzle state is set as **2**.

## **Manhattan Distance Heuristic**

This heuristic is calculated by finding the number of places the initial state of the element is away from its place in the goal state. This value is calculated for each element and then the sum of those values is set as the h(n) value for that state.

**Goal State: 1 2 3 Puzzle State: 1 2 4**

**4 5 6 3 0 6**

**7 8 0 7 8 5**

h(4)= 1 + 2 = **3**, h(3) = 1 + 2 = **3**, h(5) = 1 + 1 = **2**

Therefore h(n) for this state = 3 + 3 + 2 = **8**

# **Results**

This Java application was tested with the following 5 puzzles:

**P1: 1 4 6 P2: 4 8 0 P3: 3 5 8**

**0 2 8 6 5 7 4 2 6**

**5 7 3 3 2 1 0 1 7**

**P4: 1 2 0 P5: 5 1 3 P6: 5 1 8**

**4 5 3 8 6 0 2 4 6**

**7 8 6 2 7 4 7 3 0**

## Number of States Expanded

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Nodes expanded** | | | |
|  | **Uniform Cost Search** | **A \* with Misplaced Tiles** | **A\* with Manhattan Distance** |
| **P1** | 68290 | 5108 | 731 |
| **P2** | 178602 | 54568 | 853 |
| **P3** | 44097 | 1841 | 264 |
| **P4** | 4 | 2 | 2 |
| **P5** | 76424 | 3937 | 444 |
| **P6** | 135757 | 13901 | 2287 |

**Fig 1.1 Number of expanded Nodes for each puzzle**

From the fig. 1.1, it can be seen that, for the relatively simpler puzzle 4 the difference in the number of nodes expanded is too small to be observed on the graph, whereas with the more challenging variants (P2, P6) it can be seen just how few the expanded nodes are in Manhattan heuristic is comparison to Uniform Search and Misplaced tile.

## Maximum Number of Nodes in the Queue

|  |  |  |  |
| --- | --- | --- | --- |
| **Maximum Number of Nodes in a Queue** | | | |
|  | **Uniform Cost Search** | **A \* with Misplaced Tiles** | **A\* with Manhattan Distance** |
| **P1** | 50081 | 5021 | 655 |
| **P2** | 62126 | 44447 | 1067 |
| **P3** | 37025 | 1968 | 317 |
| **P4** | 9 | 4 | 4 |
| **P5** | 52171 | 3879 | 442 |
| **P6** | 61886 | 13260 | 2209 |

**Fig 1.2 Number of expanded Nodes for each puzzle**

From the fig. 1.2 it can be observed that, the size of the queue is much smaller in the case of the heuristic search algorithms in comparison to the uniform cost search.

# **Conclusion**

It can be seen from the result that the performance of Uniform Cost Search is the worst among the three, this is because the A\* algorithm with h(n) = 0 has a time complexity of O(b^d). Among the other heuristic search algorithms, the Manhattan distance method shows the best performance. Both the Manhattan distance and the Misplaced Tile heuristic methods greatly improve the performance from the Uniform Cost Search as is evident from the results.

# **Output Trace**

Type 1 to use a default puzzle, or 2 to enter your own puzzle.

2

0 is used to denote the blank space

Enter the numbers for Line 0:

0 1 3

Enter the numbers for Line 1:

4 2 5

Enter the numbers for Line 2:

7 8 6

Enter your choice of algorithm

1.Uniform Cost Search

2.A\* with the Misplaced Tile heuristic.

3.A\* with the Manhattan distance heuristic.

3

The best state to expand with a G(n)=0 and H(n)=4 is:

0 1 3

4 2 5

7 8 6

The best state to expand with a G(n)=1 and H(n)=3 is:

1 0 3

4 2 5

7 8 6

The best state to expand with a G(n)=2 and H(n)=2 is:

1 2 3

4 0 5

7 8 6

The best state to expand with a G(n)=3 and H(n)=1 is:

1 2 3

4 5 0

7 8 6

The best state to expand with a G(n)=4 and H(n)=0 is:

1 2 3

4 5 6

7 8 0

Success

Nodes Expanded: 4

Max Queue Size: 9

Depth of the Solution is: 5