CMSC 142 MP3: 8-Puzzle Problem

Juancho Meneses
University of the Philippines Baguio
Governor Pack Road
Baguio, Benguet 2600
jrv.meneses@gmail.com

Gimel Velasco
University of the Philippines Baguio
Governor Pack Road
Baguio, Benguet 2600
gfvelasco@up.edu.ph

ABSTRACT

This paper presents 3 different implementation of solving the 8-Puzzle using branch and bound. The three implementations consists of the Hamming function, the Manhattan function and the Breadth-first Search. To compare the performances of each implementation, the average running times of each of the functions in a particular number of trials are collected using only one particular non-disjoint tile puzzle.

1 INTRODUCTION

The 8-Puzzle is made of 9 squares fashioned in a $3x3$ manner that consists of 8 pieces of tiles that needs to be ordered in such a way that each of the 8 pieces must be placed in its proper place in the $3x3$ board. With 9 squares and 8 pieces, there will be a free space. The challenge for the 8-Puzzle is that the pieces cannot be lifted but can only moved in a "slide" action. To make an efficient approach for the puzzle, the point of view for searching for the solution to the puzzle is that the free space is considered as the "moving piece". With this, there would exist four possible actions each movement: north movement, south movement, west movement and east movement of the free space. [2]

2 METHODOLOGY

For solving the 8-Puzzles in this case, Heuristic Algorithms will be used. The algorithms' job is to find the most promising part of the search tree by making use of the information that can be gathered from the domain of the problem.

2.1 Search Tree

It was discussed above that the free space is the one considered as the "moving piece" so as to simplify and make efficient the process of searching for the solution to a particular 8-Puzzle Problem. The free space can either move north, south, east or west. For every change in the free space's place, this represents a state node in the search tree. For the search tree of the 8-Puzzle, a node in the search tree represents a possible solution and an edge represents the allowable moves of the free space. Therefore, the graph of solvable 8-Puzzles has 182440 vertices and 241920 edges. [1]

2.2 The Three 8-Puzzle Solving Algorithms

Two most commonly used heuristics for the 8-Puzzle involve counting the number of misplaced tiles (Hamming Heuristic) and finding the sum of the Manhattan distances between each block and its goal position (Manhattan Heuristic). To further explain the two heuristics, the goal state shown in Figure 1 will be used.

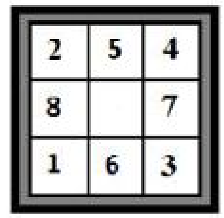

Figure 1: Sample Goal State

2.2.1 Hamming Function. The Hamming heuristic makes use of the number of misplaced tiles for computing the Hamming Function h_H. The Hamming function is computed by simply incrementing 1 in every tile that is not in the goal state. A demonstration on how to get the Hamming function is shown in Figure 2.

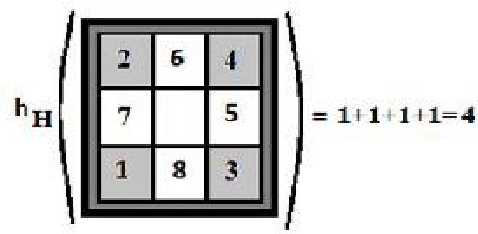

Figure 2: Hamming Function computation

The Hamming Function implemented in Java is shown at Figure 3

```java
public int numWrongTiles()
{
    int count = 0;
    for (int i=0; i < puzzle.length; i++)
    {
        for (int j=0; j < puzzle[i].length; j++)
        {
            if ((puzzle[i][j] != key[i][j]) && puzzle[i][j] != 0)
            {
                count++;
            }
        }
    }
    return count;
}
```

Figure 3: Hamming Function implementation in Java

2.2.2 Manhattan Function. The Manhattan heuristic uses the summation of each tiles' Manhattan distance for computing the Manhattan Function h_M. The Manhattan function is calculated as:

$$h_M(S) = \sum ManhattanDistance(k) \tag{1}$$

where $k \in \{1, 2, ..., 7, 8\}$ since there are 8 tiles for the 8-Puzzle. $ManhattanDistance(k)$ is then calculated as:

$$ManhattanDistance(k) = |x_k - x_{kg}| + |y_k - y_{kg}| \tag{2}$$

where x_k and x_{kg} is the horizontal distance between a particular tile k to its goal position and y_k and y_{kg} is the vertical distance between a particular tile k to its goal position. Thus (x_k, y_k) is the coordinates of a particular tile k and its goal position is (x_{kg}, y_{kg}). A demonstration on how to get the Manhattan Function is shown in Figure 4.

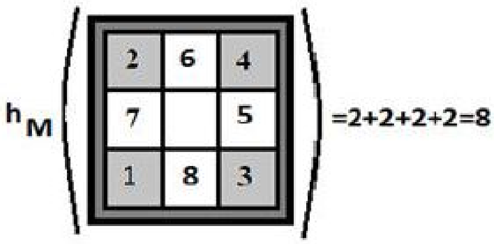

Figure 4: Manhattan Function computation

The Manhattan Function implemented in Java is shown at Figure 5

```java
public int manhattanDistance(int key)
{
    if (key < 0 || key > 9)
        return -1;

    int row = findKeyRow(key);
    int col = findKeyCol(key);

    int correctRow = -1;
    int correctCol = -1;

    if (key == 0) {
        return -1;
    }
    else if (key == 1) {
        correctRow = 0; correctCol = 0;
    }
    else if (key == 2) {
        correctRow = 0; correctCol = 1;
    }
    else if (key == 3) {
        correctRow = 0; correctCol = 2;
    }
    else if (key == 4) {
        correctRow = 1; correctCol = 0;
    }
    else if (key == 5) {
        correctRow = 1; correctCol = 1;
    }
    else if (key == 6) {
        correctRow = 1; correctCol = 2;
    }
    else if (key == 7) {
        correctRow = 2; correctCol = 0;
    }
    else if (key == 8) {
        correctRow = 2; correctCol = 1;
    }

    return Math.abs(correctRow-row) + Math.abs(correctCol-col);
}
```

Figure 5: Manhattan Function implementation in Java

2.2.3 Breadth-first Search. The Breadth First Search, unlike the two procedures stated above, is a blind search algorithm. Hence, this kind of approach does not use any heuristics since it does not behave based on the state of the problem since it simply traversing the whole search tree until it finds the goal state. This is used so that the differences of blind search and heuristics for this kind of problem would be observed.

3 RESULTS AND DISCUSSIONS

The program was run in an Asus laptop running Windows 10 Home Single Language with system specs shown below:

System Specifications

Processor	Intel(R) Celeron(R) CPU N2940 at 1.83GHz
Installed Memory(RAM)	4.00GB (3.89GB usable)
System type	64-bit Operating System, x64 based processor

After running 10 trials on each of the three 8-Puzzle solving algorithms, the runtimes were gathered and are averaged. Each trial and each solver has used only one randomized initial configuration so as to ensure a consistent and reliable data. The results yielded are as follows:

Hamming Heuristic

Trial	Runtime(ms)
1st	6149
2nd	6189
3rd	8582
4th	7038
5th	6315
6th	5467
7th	5365
8th	6427
9th	6100
10th	5650

Average Time: 6328.2ms

Manhattan Heuristic

Trial	Runtime(ms)
1st	1053
2nd	1350
3rd	1542
4th	1670
5th	1358
6th	1124
7th	1382
8th	1739
9th	1901
10th	1524

Average Time: 1464.3ms

Breadth First Search

Trial	Runtime(ms)
1st	3612
2nd	3376
3rd	3478
4th	2771
5th	2596
6th	2335
7th	2971
8th	3463
9th	3367
10th	3284

Average Time: 3125.3ms

Based on the average runtimes of the three algorithms, the Manhattan Heuristic is found to be the fastest heuristic that could solve the 8-Puzzle Problem. This is because the Manhattan heuristic is much more informed than the Hamming heuristic since the Manhattan function takes the distances into account meanwhile the Hamming function only counts the misplaced tiles. Also, the Manhattan heuristic solves the first tiles that are closer to its goal state and hence having a more efficient approach to solving the puzzle.

4 CONCLUSIONS

For solving the 8-Puzzle Problem, three heuristic algorithms were presented: The Hamming Heuristic, Manhattan Heuristic and a Self-made Algorithm. In comparing which algorithm best suits in solving the 8-Puzzle Problem, the average runtimes of 10 trials of each algorithm is gathered. It is found that the Manhattan Heuristic best suits in solving the 8-Puzzle. The Manhattan heuristic yielded an average runtime of 1464.3 milliseconds while the Hamming heuristic yielded an average runtime of 6328.2 milliseconds and the Breadth-first Search yielded an average runtime of 3125.3 milliseconds. To further investigate on the reason why the Breadth-first search is faster than the Hamming heuristic, it is advised to tackle different initial configurations of the 8-Puzzle. With this, the Manhattan heuristic was found to be 4 times faster than the Hamming heuristic and is therefore the fastest method that could find a solution to the 8-Puzzle Problem.

REFERENCES

[1] Douglas Harder. A-star Search Algorithm. web.ics.purdue.edu/~elgamala/ECE368/Slide32-AstarSearch.pptx. (????). Accessed: 2017-04-29.

[2] Anca-Elena Iordan. 2016. A Comparative Study of Three Heuristic Functions Used to Solve the 8-Puzzle. *British Journal of Mathematics and Computer Science* (April 2016).