CS440 - Artificial Intelligence
Assignment 1 - Part 2

By,
Udit Mehrotra (umehrot2)
Sanjana Chandrashekar (chndrsh4)
Suhas Hoskote Muralidhar (shmural2)

8 Puzzle

After generating 50 random puzzles of varying difficulty, we solved each puzzle with the given 3 heuristics (Misplaced tiles, Manhattan distance and Gasching's).

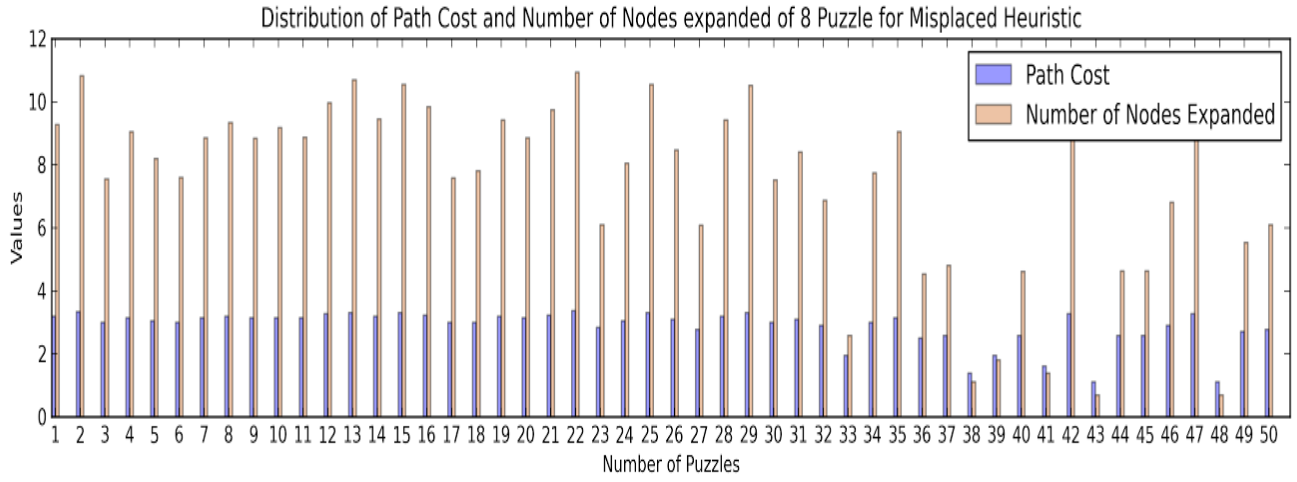
We implemented 3 modular functions to calculate individual heuristic value by passing value 1, 2 or 3 respectively. This way we are just plugging in the heuristic value and add it to path cost to find the nodes to expand in each step.

Also, since there was huge difference in values between Path Cost and Number of nodes expanded, we used log-log histogram to chart the variation between the two.

Below are the findings from each run

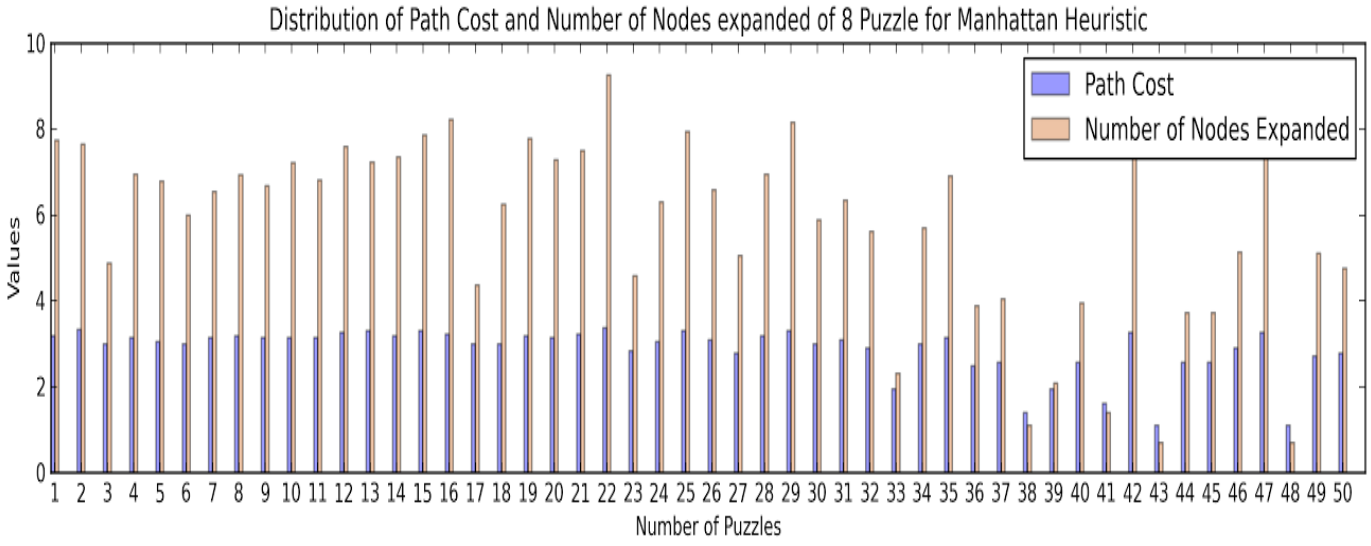
1. **Heuristic 1 :** Number of Misplaced tiles

Average Path Cost	19.46
Average number of nodes expanded	10289.42



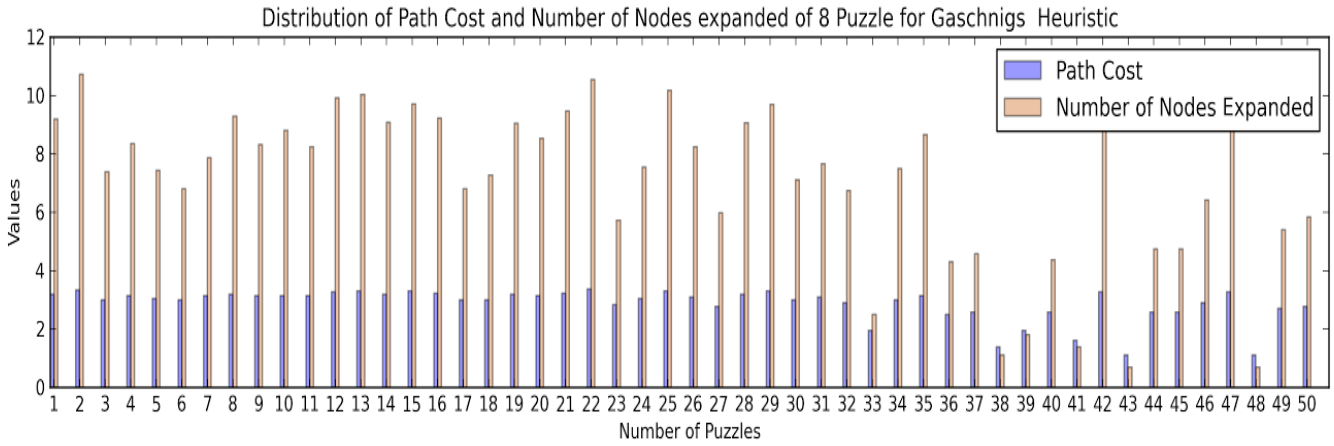
2. **Heuristic 2 :** Sum of Manhattan distances between every tile's current and desired position.

Average Path Cost	19.46
Average number of nodes expanded	1155.75



3. **Heuristic 3:** Gaschnig's heuristic

Average Path Cost	19.46
Average number of nodes expanded	6977.68



Gasching's Heuristic is calculated as follows:
Algorithm:
Input: 2D Puzzle array
Output: Number of steps taken to reach the goal state

```
steps = 0
While(input is not solution)
        if blank is in final position
                swap blank with any mismatched tile
        else
                swap blank with actual matched tile
        steps++
return steps
```

Java Code Snippet to calculate Gasching's Heuristic

```
while(!checkSolution(input, solution)){
    // if zero is not in its position
    if(zeroPosition.x != 0 || zeroPosition.y != 0){
        int correctTile = getActualTile(zeroPosition.x, zeroPosition.y);

        int correctTileX = currentCoord.get(correctTile).x;
        int correctTileY = currentCoord.get(correctTile).y;

        // swap blank with the correct value
        input[zeroPosition.x][zeroPosition.y] = input[correctTileX][correctTileY];
        input[correctTileX][correctTileY] = 0;

        // update blank tile position
        zeroPosition.x = correctTileX;
        zeroPosition.y = correctTileY;

        // remove corrected tile from misplaced queue
        misplacedTile.remove(correctTile);
        //printArrayC(input);

    }else{
        // if zero is in correct place, but puzzle is not solved yet, get first element from misplaced tile queue
        int tempMisplacedTile = misplacedTile.peek();

        // get misplaced tile's current coordinates
        int tempX = currentCoord.get(tempMisplacedTile).x;
        int tempY = currentCoord.get(tempMisplacedTile).y;

        // swap misplaced tile with blank
        input[zeroPosition.x][zeroPosition.y] = tempMisplacedTile;
        input[tempX][tempY] = 0;

        // update blank and misplaced tile position
        currentCoord.put(tempMisplacedTile, new Coord(zeroPosition.x, zeroPosition.y));
        zeroPosition.x = tempX;
        zeroPosition.y = tempY;
    }
    count++;
}
return count;
}
```

Scaling behaviour of 8 Puzzle algorithms

From Table 1 below, we can observe that as the Path cost to solve the 8 puzzle increases, in most of the cases the number of nodes expanded for each of the heuristic in increases. There are also cases where even though the path cost is more the number of nodes expanded might be less, as it depends on the ease of solving particular puzzle.

Among the 3 heuristic, for a given path cost, we observe that Misplaced Tile heuristic expands more nodes compared to the other 2, whereas sum of manhattan distance heuristic expands the least.

The above observation is expected, as for 8 puzzle problem, sum of Manhattan distance heuristic dominates misplaced tile heuristic and hence it is more effective. This results in it expanding fewer nodes compared to that of misplaced tile heuristic.

In case of Gasching's heuristic, which is a relaxed heuristic in which any tile can be moved to a blank's position, the number of nodes expanded is more than that of Manhattan distance

heuristic and less than that of Misplaced tile heuristic. This can be justified based on the relaxation of the heuristic, in which due to the flexibility of any tile moving to blank position, there is a high probability of expanding more nodes compared to Manhattan heuristic.

Table showing Path cost and Number of Nodes expanded for each of the three Heuristic
(Table 1)

sl.no	Path Cost	Nodes expanded Misplaced tile	Nodes expanded Manhattan heuristic	Nodes expanded Gasching's heuristic
1	24	10633	2288	9827
2	28	49541	2073	45027
3	20	1871	131	1591
4	23	8376	1033	4250
5	21	3601	886	1676
6	20	1974	398	896
7	23	6955	690	2593
8	24	11230	1022	10709
9	23	6800	786	4068
10	23	9571	1352	6638
11	23	7030	902	3785
12	26	21127	1981	20103
13	27	43430	1377	22606
14	24	12651	1553	8692
15	27	37609	2596	16329
16	25	18618	3693	10137

17	20	1941	78	890
18	20	2419	512	1440
19	24	12188	2374	8498
20	23	6992	1449	5068
21	25	16767	1790	12867
22	29	55718	10435	38098
23	17	442	97	302
24	21	3110	542	1891
25	27	37924	2790	25903
26	22	4691	722	3749
27	16	432	156	394
28	24	12145	1041	8530
29	27	36593	3478	16095
30	20	1818	357	1209
31	22	4431	565	2094
32	18	950	275	843
33	7	13	10	12
34	20	2273	296	1795
35	23	8494	993	5774
36	12	92	48	73
37	13	121	57	97
38	4	3	3	3
39	7	6	8	6

40	13	100	52	79
41	5	4	4	4
42	26	25415	2619	19935
43	3	2	2	2
44	13	102	41	113
45	13	102	41	113
46	18	897	169	604
47	26	26574	3740	22916
48	3	2	2	2
49	15	249	164	220
50	16	444	116	338

Execution:

1. Unzip **Muralidhar_Suhas_Hoskote_a1_part2.zip**
2. cd **Muralidhar_Suhas_Hoskote_a1_part2**
3. javac *.java
4. java EightPuzzle
5. Output: 3 CSV files with path cost and number of nodes expanded for each of the three heuristic