# Intelligent Systems

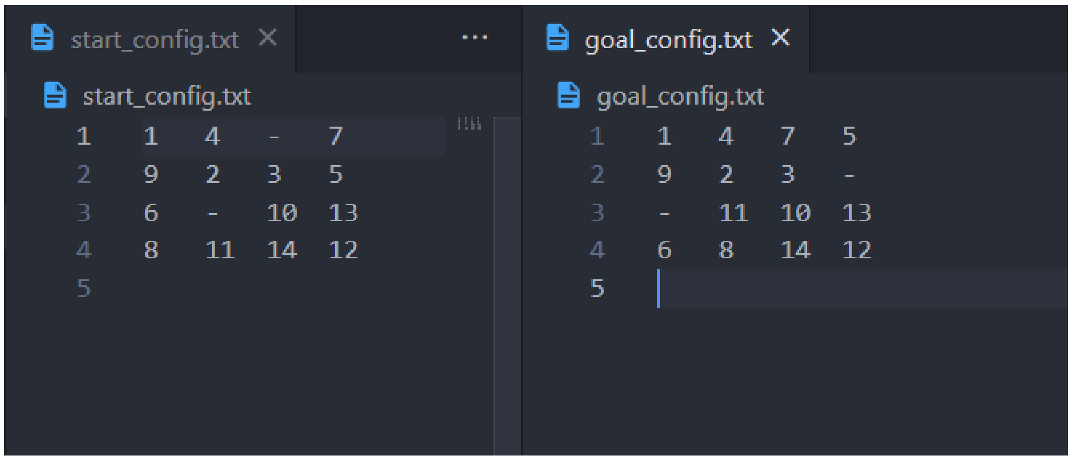
# CS3612

<u>Search: Modified n-puzzle</u>

**Reg. No: 190023G**

## Introduction

An N-puzzle involves solving a square board filled with numbered tiles. The board of the basic puzzle has simply one vacant area that may be filled with tiles to change the layout. The rearranged tiles must produce the necessary goal state for the problem to be solved. In this research I concentrated on a problem with two vacant ("-") areas rather than one. N tiles make up the problem, and there are two vacant spots for the tiles to travel between. The puzzle has Start and Goal configurations, commonly known as states. By rearranging the tiles in the empty spots, the puzzle may be solved, and the Goal configuration achieved. A* algorithm involves here. [1]

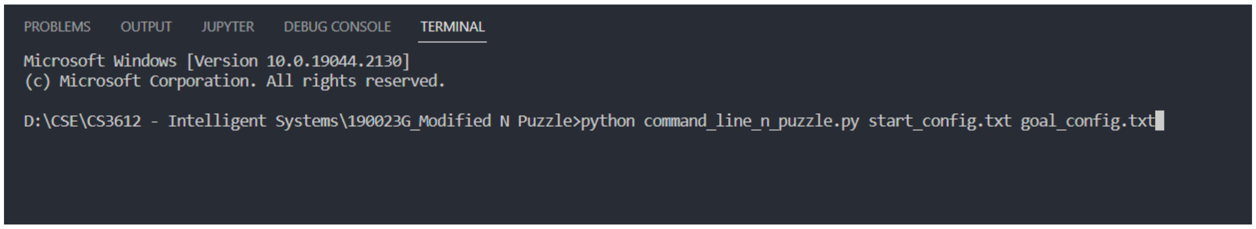


```
start_config.txt
1   4   -   7
9   2   3   5
6   -   10  13
8   11  14  12
```

```
goal_config.txt
1   4   7   5
9   2   3   -
-   11  10  13
6   8   14  12
```

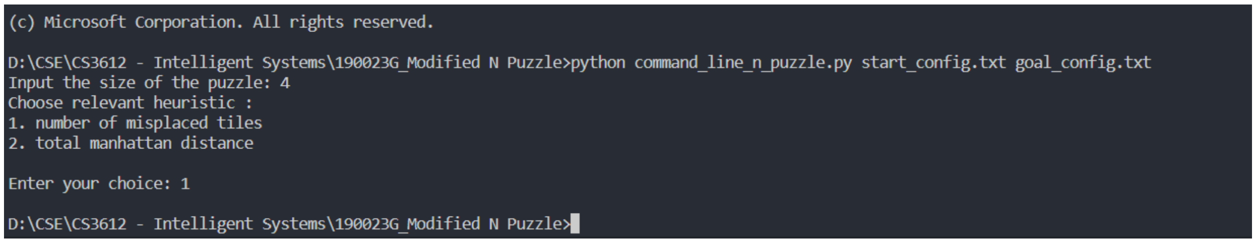
*Figure 1: Sample image of start and goal configuration*

## Run on Command Prompt



```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

D:\CSE\CS3612 - Intelligent Systems\190023G_Modified N Puzzle>python command_line_n_puzzle.py start_config.txt goal_config.txt
```

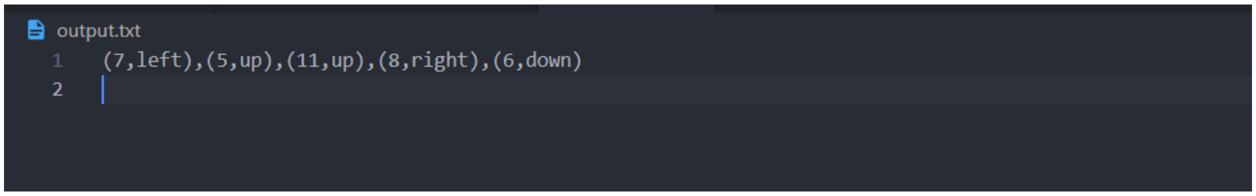
By above command line argument, can run the sample test cases given in the .txt files. And get the output in the output.txt file.



```
(c) Microsoft Corporation. All rights reserved.

D:\CSE\CS3612 - Intelligent Systems\190023G_Modified N Puzzle>python command_line_n_puzzle.py start_config.txt goal_config.txt
Input the size of the puzzle: 4
Choose relevant heuristic :
1. number of misplaced tiles
2. total manhattan distance

Enter your choice: 1

D:\CSE\CS3612 - Intelligent Systems\190023G_Modified N Puzzle>
```

*Figure 2 Sample input for command_line_n_puzzle.py*

Here, we can experiment the output for both misplaced tiles and Manhattan distance by using the relevant input.

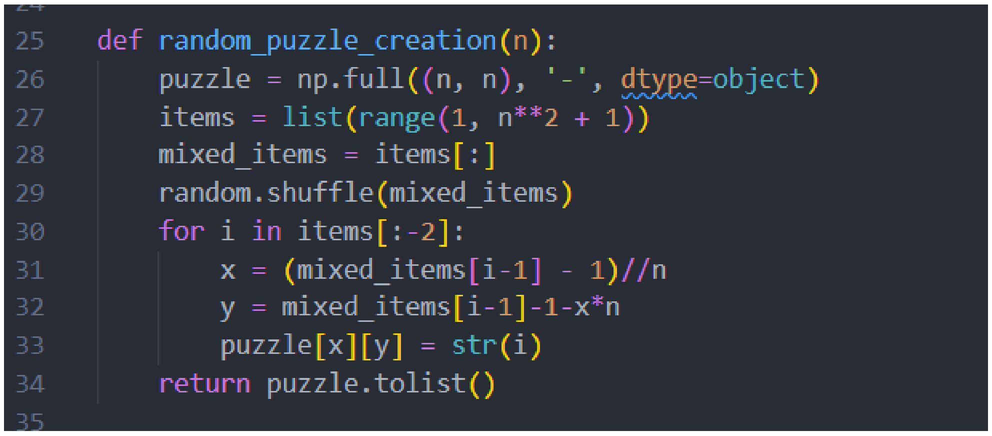


```
output.txt
(7,left),(5,up),(11,up),(8,right),(6,down)

```

*Figure 3 : Sample output stores in output.txt file*

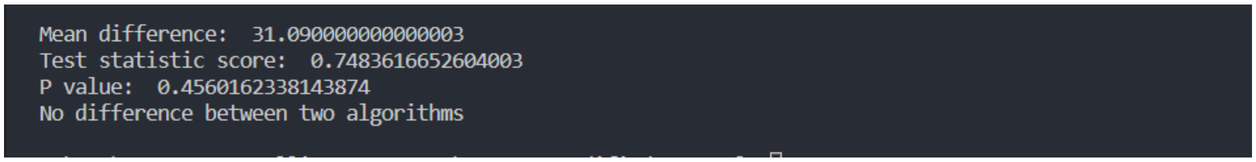
## Randomly generated test cases

The following code was used to create 100 n-puzzle problems at random with puzzle sizes ranging from 5 to 20. Which is stored in random_tests.py



```python
def random_puzzle_creation(n):
    puzzle = np.full((n, n), '-', dtype=object)
    items = list(range(1, n**2 + 1))
    mixed_items = items[:]
    random.shuffle(mixed_items)
    for i in items[:-2]:
        x = (mixed_items[i-1] - 1)//n
        y = mixed_items[i-1]-1-x*n
        puzzle[x][y] = str(i)
    return puzzle.tolist()
```

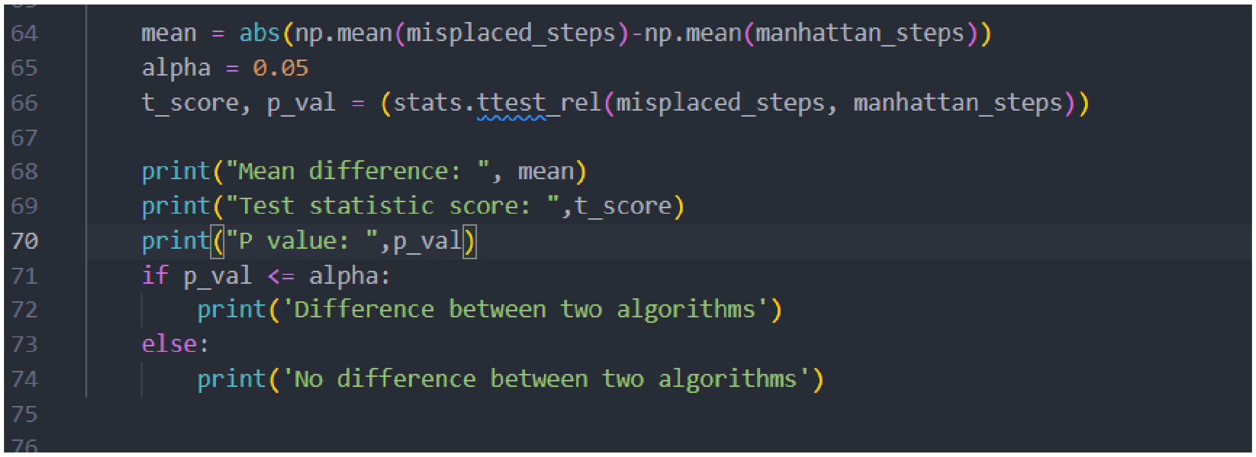
The output displayed below demonstrates the creation of 100 start and goal pairings as well as their statistical analysis following the application of the two heuristics. misplaced tiles, Manhattan Distance



```
Mean difference:  31.090000000000003
Test statistic score:  0.7483616652604003
P value:  0.4560162338143874
No difference between two algorithms
```

According to the analysis above, a significant value (p-value) of roughly 0.46 is achieved for the specific 100 randomly generated start and goal pairs. A p-value of 0.05 or less would be required to reject the null hypothesis at a 95% confidence level. Since there is no significant difference between the two implementations employing the two heuristics in this case. The statistical analysis uses the Scipy and Numpy libraries.

Following code part used for analysis



```
    mean = abs(np.mean(misplaced_steps)-np.mean(manhattan_steps))
    alpha = 0.05
    t_score, p_val = (stats.ttest_rel(misplaced_steps, manhattan_steps))

    print("Mean difference: ", mean)
    print("Test statistic score: ",t_score)
    print("P value: ",p_val)
    if p_val <= alpha:
        print('Difference between two algorithms')
    else:
        print('No difference between two algorithms')
```

## References

[1] “A-star(A*) in general”

https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/