Application of Search Algorithms on 8-Puzzle

Application of BFS, UCS, DFS, IDS, GBS, A on n-puzzle*

Barışcan Tunalı
Big Data Analytics and Applications
Bahçeşehir University
Istanbul, Turkey
bariscantunali@outlook.com

***Abstract*— This paper is about comparison results between search algorithms based on their performance on solving n-puzzle problem. To compare those algorithms base generator and solver classes was created so every algorithm took same matrix everytime.**

Keywords—BFS;UCS;DFS;IDS;GBS;A*;Search;Algorithm

I. INTRODUCTION

The main reason we are doing this study is comparing efficiency of search algorithms on np-hard problems. With this study, testing different algorithms on same puzzle will show us which one is more efficient on n-puzzle problem respectively. My basic assumptions was opening leafs on clockwise and there will be no repetitive leafs among all queue to decrease processing time and memory size. Although we compare new nodes with list, this adds extra processing time but this can be ignored.

II. EXPERIMENTAL SETUP

I used .NET framework 4.5.2, c# and visual studio 2015 as platform, language and IDE. I created several classes to perform experiment and get output more efficiently. Swapper functions, Neighbor finding functions depending on our algorithm. And Completeness control function to determine is this matrix complete or not. And I also have item classes, because generating matrix was not enough. I needed to control is complete? , is expanded? Or is checked? For each Matrix. So I put these attributes into class along with Matrix itself, level, matrix's ID and parent ID. And finally toString functions, I modified toString functions to print matrix itself (if necessary), and print memory size, time, node count, max level, and is Complete. I needed is complete because starting from 8 puzzle personal computers even if they have memory enough doesn't have enough processor power nor time to calculate all of these combinations so I had to limit level size with 11, if program can't solve before 11^{th} level it stops and starts next iteration. I created puzzle generator class which takes length of a size which is determined by user, and generates random n,n puzzle. As I mentioned before I created simple user interface, Program user determines length of a square himself. With this option my program can create 8 puzzle, 15 puzzle or n-puzzle.

III. EXPERIMENTAL RESULTS

BFS, DFS, UCS couldn't find solution in acceptable time so after limiting levels. Some of results are attached below.
Size : 7414884 byte,Node Count : 205969,Max Level : 11,Time : 4380885 Miliseconds

Size : 3707460 byte,Node Count : 102985,Max Level : 11,Time : 1062723 Miliseconds

Size : 3707460 byte,Node Count : 102985,Max Level : 11,Time : 1068224 Miliseconds

Size : 5392656 byte,Node Count : 149796,Max Level : 11,Time : 2175918 Miliseconds

Size : 2630232 byte,Node Count : 73062,Max Level : 11,Time : 478347 Miliseconds

In the function of DFS I limited depth by 10000 level, because n-puzzle is np-hard problem, DFS can go deeper forefer. And possibly cannot find the solution. My DFS assumption was, tree will expand always clockwise.

In the function of UCS my cost formula was this,

```
temp+=Math.Abs(Matrix[i,j]-((i*sideLenght)+j));
```

With this formula I get current cell's value, subtract sum of x and y coordinates, get absolute value of this value. Add it to total score. With this formula I calculate each leaf's score and assign it to Cost attribute at class. After I add all leafs, I sort queue by this cost. With this I always pop lowest cost from queue. Why did I used this formula, let's consider 0,0 coordinates in array. According to my implementation there should be zero so 0 – (0*3+0) = 0, Or (1,2) = 5 – (1*3+2) = 0, So we see at solution case all our total cost is zero, but otherwise it's not. But this formula also has a side effect or flaw which is if our array is like 8,1,2,3,4,5,6,7,0. In here our

total cost is 16, but since most distant corners needs to swap and it’s not possible on n-puzzle. Actual cost is lot more.

Results for DFS :

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8062 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8058 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8570 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8714 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8763 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 9394 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8503 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8491 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8587 Milisaniye

Size : 360036 byte
Node Count : 10001
Max Level : 10000
Is Complete : No
DFS - Time : 8608 Milisaniye

Results for UCS :

Size : 463428 byte
Node Count : 12873
Max Level : 9
Is Complete : No
UCS - Time : 25136 Milisaniye

Size : 463428 byte
Node Count : 12873
Max Level : 9
Is Complete : No
UCS - Time : 22818 Milisaniye

Size : 674064 byte
Node Count : 18724
Max Level : 9
Is Complete : No
UCS - Time : 44851 Milisaniye

Size : 674064 byte
Node Count : 18724
Max Level : 9
Is Complete : No
UCS - Time : 44662 Milisaniye

Size : 674064 byte
Node Count : 18724
Max Level : 9
Is Complete : No
UCS - Time : 43589 Milisaniye

Size : 674064 byte
Node Count : 18724
Max Level : 9
Is Complete : No
UCS - Time : 45847 Milisaniye

Size : 463428 byte
Node Count : 12873
Max Level : 9
Is Complete : No
UCS - Time : 20792 Milisaniye

Size : 463428 byte
Node Count : 12873
Max Level : 9
Is Complete : No
UCS - Time : 21334 Milisaniye

Size : 674064 byte
Node Count : 18724
Max Level : 9
Is Complete : No
UCS - Time : 44228 Milisaniye

Size : 926820 byte
Node Count : 25745
Max Level : 9
Is Complete : No
UCS - Time : 86446 Milisaniye

IV. DISCUSSION

As expected BFS, DFS did not find the solution on given limit. UCS also couldn't find even though given costs . UCS on n-puzzle doesn't narrow space consumption but increases problem solvability with priority queue. But sorting and calculating cost adds an extra time complexity. We understand from here, Algorithms like BFS or DFS which search the problem space uninformed, brute-force, is more likely can't find a solution in acceptable time range. So we need an informed search algorithm like A*, or heuristic method like Manhattan distance etc. Even informed search algorithms doesn't guarantee solution for np-hard problems but merely narrows time and/or space complexity.

V. CONCLUSION

Since this is home work, my main limitation was my computer, this kind of algorithms is very dependent on coding platform, method. E.g. Visual studio doesn't allow program to use multiple core it runs only on a core. My methods was on nxn integer arrays but solutions that I researched, was based on one dimensional string arrays.