Report
VE593
Jia Shi
Oct.22.2018

Q1:

Please note that if you are running nPuzzle, use heuristic_1, SimpleValuedGraph for heuristic_2

I would not be able to get any result for n=5. I tried to use the fastest algorithm Astar to run that, but still running after 1.5hours, which only take 0.03seconds to run n=3.

All the following case are for n=3

I wrote a method which can generate valid test case by start from the the correct state and randomly move it for 200 times. I use 20 randomly generated test case to test the computation time for each algorithm.

For BFS, the average running time is around 3 second. Most time under 2s, but still sometime have 7or 8 seconds, but never excel 10 seconds. This is the fastest and most stable one in the regular graph search(not value and beside Astar)

DFS is highly depend on the situation, most of the time it is under 5 seconds, even below 1 seconds, however, some case may last for 30 seconds, which is high depend on the case. This is because of it traits of 'going as deep as possible'. Average is 13 seconds.

DLS is kind of like the DFS, which also depend on my setting of depth limit. The running time is highly depend on the case and the limit. Depth limit of 500 will have a average running time of 23s, but limit of 100 would only have a average of 5 seconds. The general average time is 10 seconds.

IDS is obviously the slowest one because it is the summation version of DLS with depth limit increase from 1 to n. Average time is 40 seconds, and pretty stable. Worth to mention that must depth limit that found the result is around 22 to 25.

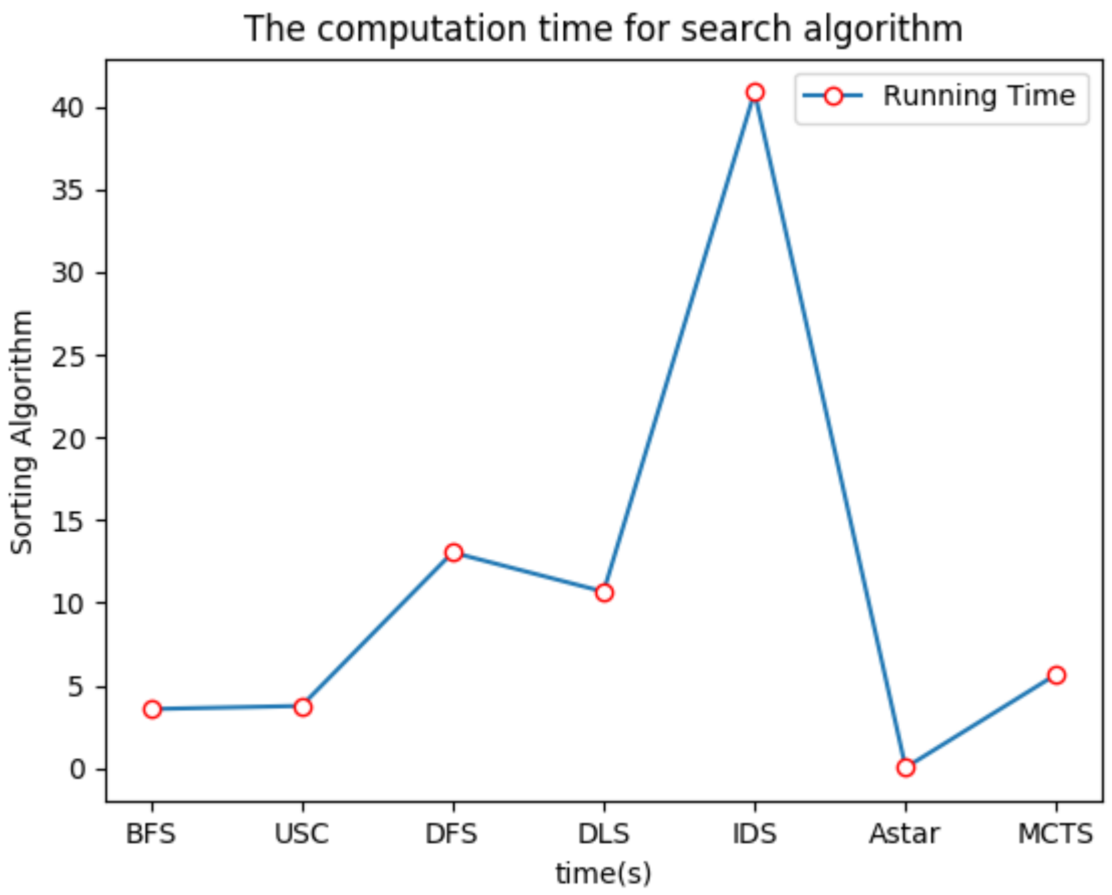
Astar, have the average running time of 0.03 seconds, is obviously the fastest searching algorithm. And it's pretty stable. Even at the worst case, the running time never exceed 0.2 seconds. However, it would take extra step to reach the goal compared to other algorithm list above.

For Monte Carlo Tree Search (MCTS), the average ruling time of it is around 5 seconds, and the time also highly depend on my setting of how many time to repeat the random search from one node. The current value is generate a random path from one single node for 50 times, then average then and compare their UCB value. Also, there's still sometime that the MCTS fall to get a result due to the setting of depth it goes. Under such situation, just run the program for several times and it will work. It's worth to mention, even though the method is pretty fast, the node it need to pass to reach the goal is relatively high, like 70 nodes in one path.
What is more, the MCTS I implemented would not run simplevaluegragh in a random way(because sometime it only have one successor, when the random index is 0,

In the graph I shown, the average running time for each algorithm is :

BFS:	3.57
UCS:	3.76
DFS:	13.04
DLS:	10.67
IDS:	40.89

Astar:0.03
MCTS: 5.68



Q2:
For this part, I implemented several methods in my clickomania state. First, I use only one dimension list to mimic a matrix of 2D, and use number from 1-n to represent different color. Then, after the display is settle done, I have one important method call 'canerase', which is to find the adjacent block with number of more than 2 in the list and then erase them in a random order. After erasing one adjacent block, I call succ to update the entire display of graph in order to realize the function of 'dropping' or 'move left or right' when the game is reaching the end. When there's no block to erase anymore(reach the end stage), I return the final score minus the cost score according to the project description.
And for the player part, I use DFS to simulation the playing process because this game won't be stuck in a 'deep loop' and will reach the end of the game quickly. Thus, I found that using DFS is efficiently because the process of playing is kind of like 'going deep' in one straight road.

PS: if the score reach 0, please rerun it, it happens sometimes