Assignment 1: search

Antti Partanen 295967

Introduction

I used python 3 to solve this problem and the code requires the following nonstandard libraries:

- numpy
  - pip3 install numpy (If you don't have it installed)
- distance
  - pip3 install distance (If you don't have it installed)

The code consists of two files: eightpuzzlesolver.py and eightpuzzlesolver_test.py.
You can run eightpuzzlesolver.py by itself, it'll print out results in one of the following forms:

- True, [iteration, running_time, len(self.visited), len(self.unvisited), len(backtrace)]
- False, [iteration, running_time]

Where True/False denotes whether the run terminated in the given iteration limits.

You can also uncomment the print statements from the code and get more detailed description. Additionally, if you give print_trace=True to the EightPuzzleSolver class as a parameter, the solver will print the steps to reach the goal form the initial state.

eightpuzzlesolver_test.py will generate csv file from the tests that it runs. It'll run every single algorithm: a*, wa* (weights: 5, 10, 15, 20) and greedy best first with every possible tile amounts. For every tile variation, the tester runs 10 random solvers with the given parameters and calculates their average statistics.

The tester runs 6 parallel processes to decrease the execution time.

The code folder also includes an excel file of the tester output for your convenience.

Results

A* (weight = 1)

	avg(iterations)	avg(running time)	avg(visited nodes)	avg(unexplored nodes)	avg(steps to solution)	successful runs
1 tile(s)	3.06	0.0009232000000000001	4.02	3.00	3.06	10
2 tile(s)	5.03	0.0023028	5.09	14.00	4.02	10
3 tile(s)	14.03	0.008781200000000001	13.05	41.06.00	6.03	10
4 tile(s)	50.09.00	0.06090990000000002	37.00.00	120.08.00	8.02	10
5 tile(s)	466.01.00	1.1749630000000002	286.02.00	745.03.00	12.02	10
6 tile(s)	651.06.00	1.1011073000000002	424.01.00	1049.02.00	14.09	10
7 tile(s)	559.01.00	0.9171999	388.02.00	732.02.00	17.05	10
8 tile(s)	1068.06.00	1.3654819999999994	978.01.00	621.00.00	22.03	10

WA* (weight = 10)

	avg(iterations)	avg(running time)	avg(visited nodes)	avg(unexplored nodes)	avg(steps to solution)	successful runs
1 tile(s)	2.06	0.0006988000000000001	3.06	2.08	3.06	10
2 tile(s)	3.09	0.0016853	4.08	12.00	4.02	10
3 tile(s)	5.07	0.0028753	6.07	20.09	6.05	10
4 tile(s)	9.07	0.006214499999999996	10.05	38.08.00	9.00	10
5 tile(s)	42.04.00	0.03727799999999999	31.00.00	100.02.00	14.02	10
6 tile(s)	68.03.00	0.0665343	37.07.00	114.09.00	18.05	10
7 tile(s)	232.05.00	0.2683156000000001	102.07.00	212.00.00	25.05.00	10
8 tile(s)	269.09.00	0.18108349999999995	211.04.00	160.02.00	33.05.00	10

Greedy Best First

	avg(iterations)	avg(running time)	avg(visited nodes)	avg(unexplored nodes)	avg(steps to solution)	successful runs
1 tile(s)	2.06	0.0007268999999999997	3.06	2.08	3.06	10
2 tile(s)	3.08	0.0015235000000000003	4.08	11.08	4.04	10
3 tile(s)	5.07	0.002821099999999999	6.07	20.09	6.05	10
4 tile(s)	9.02	0.005451000000000003	10.02	38.02.00	9.02	10
5 tile(s)	17.166666666666668	0.007285333333333366	18.00	71.33333333333333	15.666666666666666	6
6 tile(s)	17.08	0.006402600000000191	18.08	64.00.00	16.02	5
7 tile(s)	22.666666666666668	0.006468333333334186	23.666666666666668	56.333333333333336	18.333333333333332	3
8 tile(s)	31.00.00	0.0042205000000006265	32.00.00	27.00.00	25.05.00	2

The a_star_stats_excel.xlsx contains the other WA* runs.

WA* with weight = 10 results in the best steps to running time combo. A* is the best considering the shortness of the total steps. Greedy best first on the other hand results in poor performance when the number of tiles is larger, given the scarcity of successful runs, i.e. the runs that terminated in the given iteration limits.

It seems that greedy best first tends to open a lot of nodes, but cannot differentiate between better and worse ones because it does not care about the spent moves. Occasionally it gets lucky and finds the solution quickly.