n-Queens Heuristic Analysis

Robert Kim

Artificial Intelligence 1

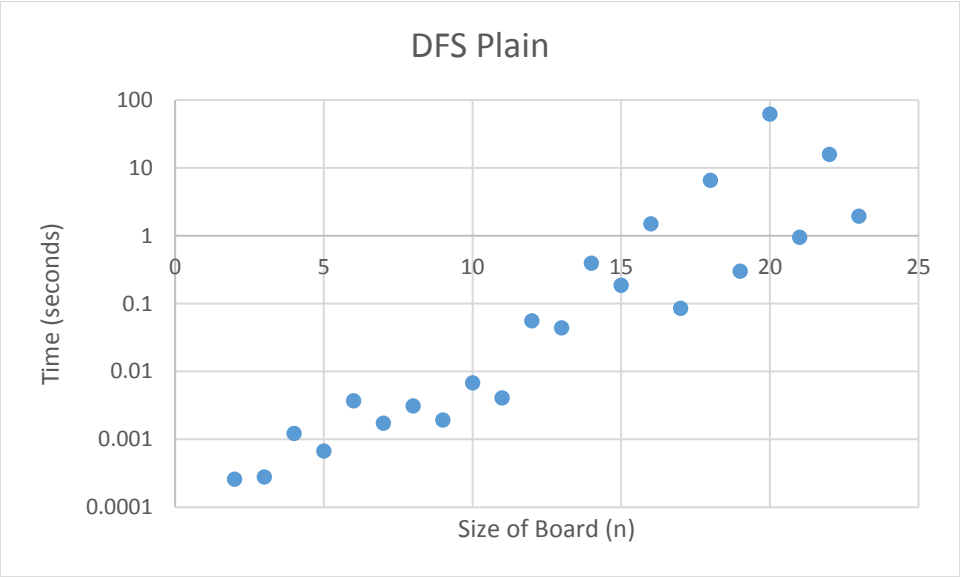
TJHSST Fall 2016

I solved the nQueens problem as a Constraint Search Problem – the program searched until it found a "board" that met specific criteria. To create this board, I used a list to track the locations of a queen in a given column (list index), and a list of sets, which tracked the available spots at a given column (list index).

At first, my code could solve n = 22 queens in a little over a minute. My best implementation can solve n = 96 queens in under 2 seconds.

Code was implemented in Python 3.5.2 and tests were performed on a Dell Inspiron 13-7368 with a dual core 2.30 GHz i5.

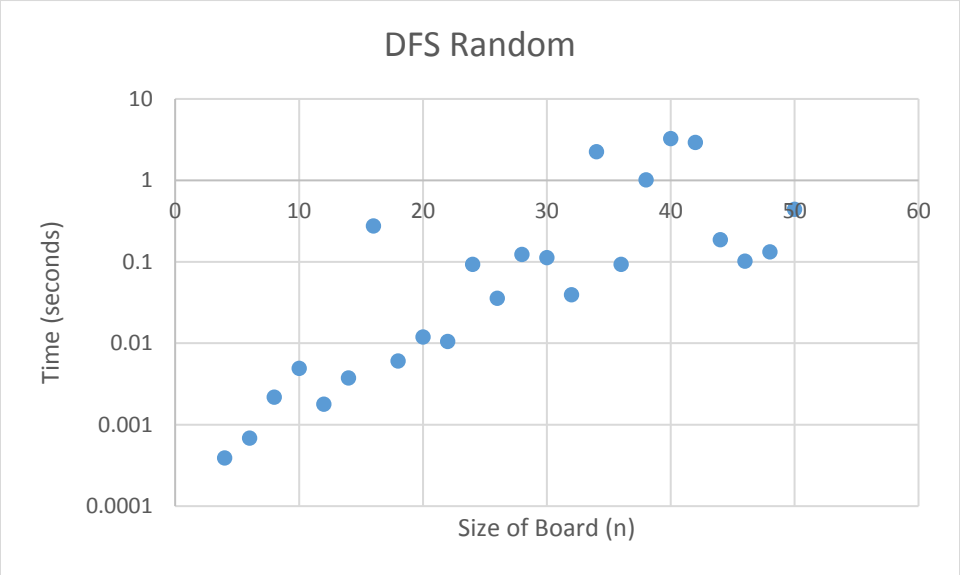
DFS Plain:



N	Goals	Nodes Created	Time (s)	Nodes/Sec
4	9	10	0.00026	38461.54
8	114	124	0.00369	33604.34
9	42	60	0.00173	34682.08
11	56	85	0.00192	44270.83
15	1377	1431	0.04386	32626.54
16	10058	10116	0.39366	25697.3
21	8563	8676	0.30011	28909.4
24	410185	410339	15.72983	26086.68
25	48525	48698	1.94928	24982.56

DFS Random Column:

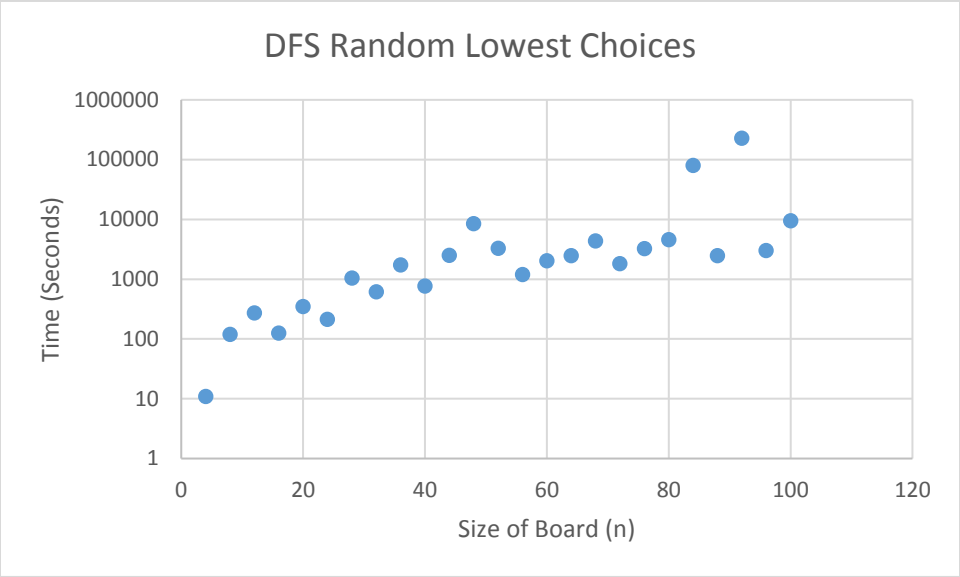
My first heuristic chose a random column of all the columns that had choices left.



N	Goals	Nodes Created	Time	Nodes/Sec
4	12	14	0.00039	35897.44
7	19	26	0.00068	38235.29
13	23	67	0.00178	37640.45
16	8086	8152	0.27468	29678.17
22	79	229	0.01047	21872.02
28	2067	2308	0.1226	18825.45
37	802	1224	0.09298	13164.12
46	136	826	0.10192	8104.396
49	3572	4403	0.43827	10046.32

DFS Lowest Choices #1:

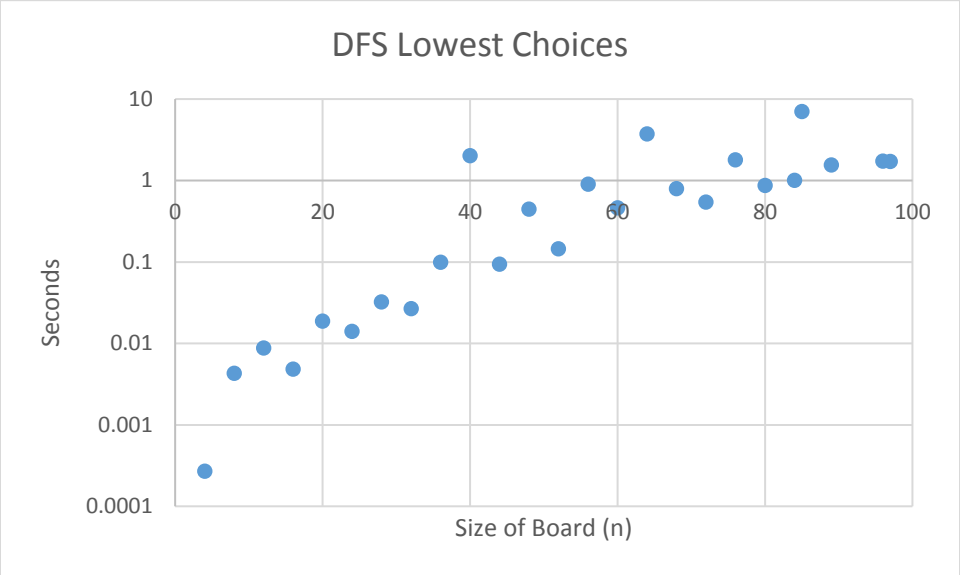
My second heuristic improved on the first one, by limiting the set of columns to those that had the lowest number of choices. Essentially, it chooses the column with the lowest number of choices and chooses a random column if there is a tie. Although using this heuristic may seem slower than the previous heuristic (Random columns), using the previous heuristic was wildly inconsistent past values of n =50, leading to >5-minute run times occasionally for some values of n. This heuristic was able to run past that, and you can see in the graph, the inconsistency of the random number generator is apparent past n = 80.



N	Goals	Nodes Created	Time	Nodes/Sec
4	10	11	0.00038	28947.37
16	63	125	0.00567	22045.86
20	250	349	0.01771	19706.38
25	59	214	0.0136	15735.29
34	1386	1744	0.19244	9062.565
40	314	771	0.09682	7963.231
45	1937	2494	0.37152	6712.963
54	278	1191	0.26369	4516.667
60	984	2041	0.43141	4730.998
65	1271	2493	0.64103	3889.054
74	1501	3252	1.16873	2782.508
80	2625	4564	1.7906	2548.866
85	77444	79631	28.65816	2778.65
94	123	3003	1.6958	1770.846
100	6443	9524	5.0344	1891.785

DFS Lowest Choices #2:

My final heuristic improved on the previous one- I decided to remove the random element of the algorithm, and instead chose the left-most column in the event of a tie.



N	Goals	Nodes Created	Time	Nodes/Sec
4	10	11	0.00027	40740.74
8	110	119	0.0043	27674.42
12	249	279	0.00879	31740.61
48	2324	2993	0.44644	6704.148
52	87	872	0.14563	5987.777
56	2640	3560	0.90804	3920.532
85	18662	20851	7.07016	2949.155
89	645	3068	1.55785	1969.381
96	147	2993	1.74171	1718.426
97	98	3005	1.71638	1750.778