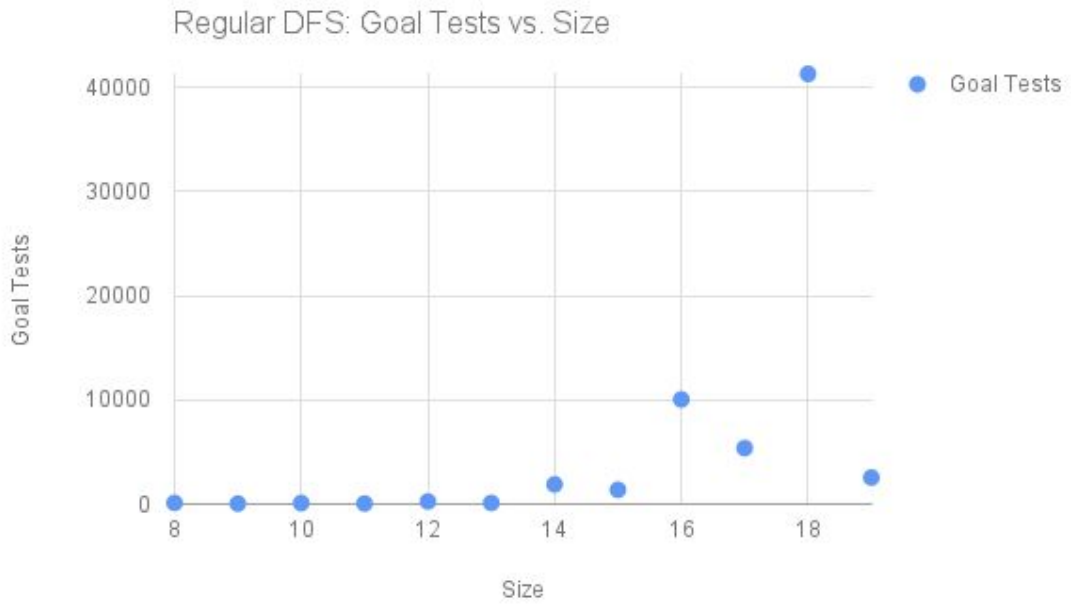
n-Queens Heuristic Analysis
Ankur Mishra
Artificial Intelligence 1
TJHSST 2016 Fall

I solved the nQueens problem as a search problem by implementing a DFS algorithm. The variables of the CSP are the n columns of a chessboard, and the values of each variable are the n rows. Initially my code could solve n=30 queens in under 2 minutes. But my best implementation can solve n = 100 queens in under a minute. I did all my programming on MacBook Pro 2010 with an 2.4Ghz i5 processor.
Case 1: Regular DFS

Table for Case 1:

Size	Goal Tests	Nodes Generated	Nodes Per Sec	Time (secs)
8	114	124	11173.03693	0.01109814644
9	42	60	10442.25062	0.005745887756
10	99	121	10126.31757	0.01194906235
11	53	83	6954.338521	0.01193499565
12	249	284	6885.168438	0.04124808311
13	112	154	4193.459862	0.03672385216
14	1889	1934	10248.8028	0.1887049675
15	1373	1428	10573.53963	0.1350541115
16	10047	10106	10463.61567	0.9658229351
17	5375	5449	10348.33454	0.5265581608
18	41299	41376	9722.425855	4.255728006
19	2546	2642	8187.781129	0.3226759434

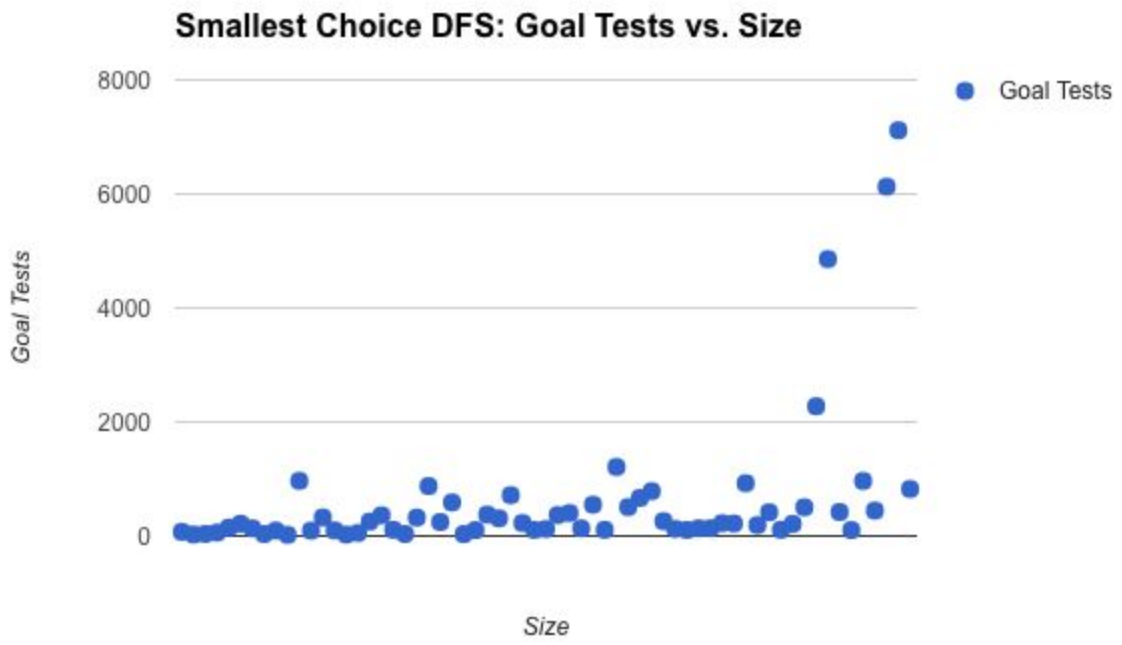


Case 2: DFS Choosing Smallest Choice

For my first heuristic I made the algorithm look for the smallest column. It drastically reduced the timings and greatly the number of goal tests, keeping it far more consistent that regular DFS.

Table for Case 2

Size	Goal Tests	Nodes Generated	Nodes Per Sec	Time
8	73	82	12184.10543	0.006730079651
20	1297	1419	5027.420998	0.2822520733
25	363	527	5236.173326	0.100646019
35	309	643	3718.887854	0.1729011536
63	4858	6021	1813.123273	3.320789099
64	421	1619	1419.806809	1.140295982
65	103	1382	1312.879732	1.052647829
66	965	2282	1413.452134	1.614486933
67	445	1787	1238.001689	1.443455219
68	6128	7506	1651.903922	4.543847799
69	7117	8527	1611.088477	5.292695045
70	825	2299	1236.844721	1.858762026

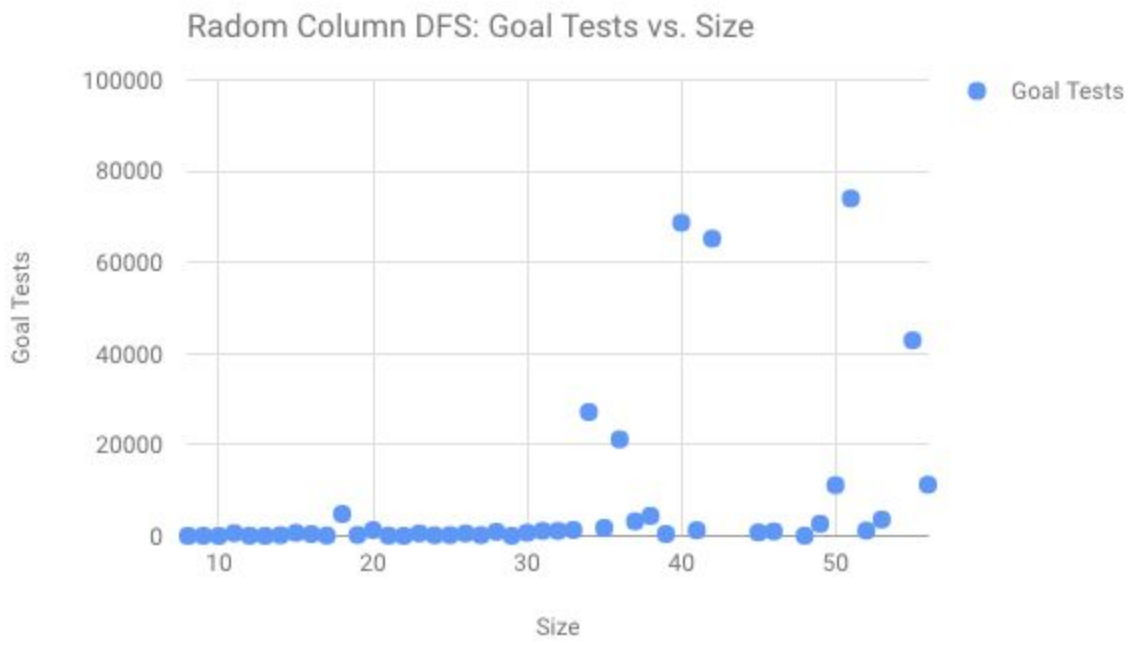


Case 3: Random Column DFS

For my second variation of DFS, I chose random column to see if it would impact my data. It actually was the fastest for smaller data sizes, but drastically got slower after hitting the data size of 30. It also had the most outliers, due to the fact that it is a random algorithm. I would be interested to see if how the closest to the middle heuristic would impact timing to the algorithm, but due to time constraints, I was unable to run it.

Table for Case 3

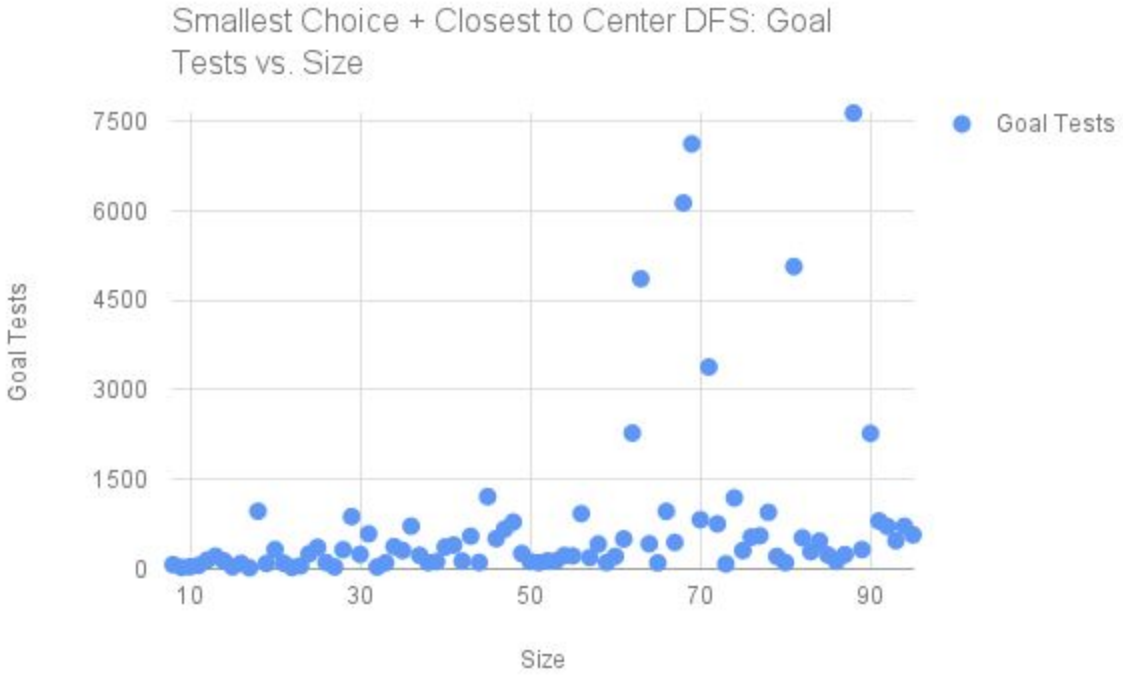
Size	Goal Tests	Nodes Generated	Nodes Per Sec	Time
8	23	39	9445.53967	0.004128932953
20	1297	1419	5027.420998	0.2822520733
25	363	527	5236.173326	0.100646019
40	68700	69222	2545.242488	27.19662285
53	3605	4532	1498.465691	3.024426937
55	42913	43900	1317.751642	33.31432009
56	11257	12293	1279.542503	9.607340097



Case 4: Smallest Column and Closest Position to the Center DFS
This variation was the fastest on average, handling 100 queens in around 6 seconds. It combined the heuristic from earlier with another one, which finds the position closest to the center. This would would make bigger constraints, speeding up the algorithm.

Table for Case 4

Size	Goal Tests	Nodes Generated	Nodes Per Sec	Time
8	73	82	21579.428284	0.00379991531
20	325	423	8117.004955	0.05211281776
25	363	527	6351.46347	0.08297300339
35	309	643	4368.684492	0.1471838951
43	549	1057	3085.959791	0.3425190449
44	109	666	2511.870945	0.2651410103
45	1211	1779	3103.303538	0.5732600689
46	507	1086	2705.065699	0.4014689922
93	473	3148	608.3104735	5.174988985
94	716	3413	567.6613028	6.012387991
95	571	3348	567.5326578	5.89921999


Smallest Choice + Closest to Center DFS: Goal Tests vs. Size
Goal Tests
7500
6000
4500
3000
1500
0
10
30
50
70
90
Size
Goal Tests