Assignment 2 Writeup

I had initially assumed when making the assignment that I needed to average 100 runs, like last assignment. Because of this, for Hill Climbing and Forward Checking, I did every run 100 times and averaged it out. However, once I learned this, I redid the table for British Museum so that I could get larger values for comparison.

DFS with Backtracking:

I only simulated through n=21 before becoming too impatient. Interestingly, the algorithm performs much better for odd ns than even ones, presumably because that's when there is a valid solution with a queen in the starting corner.

n	number_of_iterations	number_of_moves	time
10	1068	1940	0.0005248928070068359
11	559	1023	0.000260462760925293
12	3316	6120	0.0016379857063293457
13	1464	2717	0.0007703089714050293
14	28381	52976	0.014922852516174317
15	21625	40545	0.011211819648742676
16	170749	321408	0.09376379251480102
17	96580	182427	0.05374171733856201
18	784511	1486440	0.5060290265083313
19	50711	96349	0.03215226650238037
20	4192126	7985000	2.8517344403266907
21	188134	359163	0.3627978038787842

British Museum:

I made this table only getting a single value (like what you apparently wanted all along) because it takes too long. I also started it at 8 to get more than 2 values.

It turns out that British Museum is a truly terrible algorithm. No wonder Mr. Smithers wasn't getting the results he wanted. I considered making it more efficient (by choosing random permutations, so that no obviously invalid sets would be chosen) before deciding that it was a waste of time.

n	number_of_iterations	number_of_moves	time (seconds)
8	109606	876848	0.678992748260498
9	500717	4506453	3.2337429523468018
10	18614104	186141040	128.13617777824402
11	28882861	317711471	212.1617078781128

Stochastic Hill Climbing:

My Hill Climbing algorithm begins with a random board. In each iteration, it checks each queen and determines which queen has the largest number of conflicts with other queens (I didn't count those as iterations because you said on Piazza that iterations that involved ruling out where not to place a queen don't count). I then checked where in that queen's row I could move it that would minimize its number of conflicts (also not counted as iterations for the same reason). In both cases, whenever I found a tie, I flipped a coin to decide which to follow. Then I moved the queen and rinsed and repeated.

When moving the queen didn't actually reduce the number of conflicts, I increased a plateau count. If that plateau count reached a certain limit (I chose the side-length of the board), I restarted the problem at a random board.

Below, I simulated from n=10 to n=40. After that, I decided to keep simulating every 10 queens to see how good my algorithm was. I got both working above n=40.

n	number_of_iterations	number_of_moves	time (seconds)
10	103.57	261.06	0.003211996555
11	133.83	338.83	0.00496216774
12	133.98	340.96	0.005470376015
13	161.72	413.07	0.00747348547
14	167.76	430.12	0.008370292187
15	177.8	456.82	0.009740693569
16	164.39	424.08	0.01032211065
17	176.44	455.43	0.01245134592
18	176.16	456.24	0.01363281488
19	212.14	550.46	0.01721746922

20	240.81	625.42	0.02086284876
21	211.28	551.47	0.02145950794
22	249.8	651	0.02581362247
23	215.91	563.59	0.02421277761
24	233	610.08	0.02755759239
25	260.46	682.45	0.03307131767
26	241.84	633.6	0.03250417471
27	264.33	693.66	0.03793428421
28	254.16	667	0.03929307699
29	239.41	630.18	0.03848743439
30	245.43	647.06	0.04239224195
31	286.68	756.09	0.05191093206
32	270.69	713.98	0.05178930521
33	339.23	893.15	0.06867338419
34	260.39	686.46	0.05586477995
35	326.72	863.96	0.0715673399
36	301.41	797.34	0.07007362604
37	288.91	764.82	0.07013406038
38	340.45	899.94	0.08712991238
39	330.01	873.36	0.08849458694
40	346.11	916.7	0.09717860937
50	387.77	1031.78	0.1654944873
60	543.84	1448.18	0.3568630552
70	531.19	1420.1	0.488884604
80	529.64	1419.2	0.6654655266
90	566.36	1518.8	0.8740437603
100	633.55	1703.16	1.18258172

For Forward Checking:

My Forward Checking algorithm uses a lookahead that keeps track of which queens we have already used or ruled out. The lookahead contains a set of all rows that a queen could be in for each column. In each iteration, we remove a random column for our row of the lookahead and place a queen there. We then move through all future columns and remove all possibilities that the queen conflicts with. If we ever find a column with no available rows, we backtrack.

When we remove a conflicted row, we add that move to a stack frame. Whenever we backtrack and remove the queen that banned that move, we pop the stack frame and add all its moves back to the lookahead. Additionally, whenever we remove a row from the lookahead to try placing a queen there, we add that move to the *previous* queen's stack frame. That way, we won't try to place another queen there in this branch of the tree, but when we backtrack through its parent, we can place another queen there in another branch of the tree.

If we ever successfully place a queen in the last row, we have an answer, and return it.

Originally, my algorithm had the same problem that DFS has, where even numbers take much longer than odd numbers, because we can't place queens in corners. To fix that, rather than thinking of a fancy ordering heuristic, I decided to randomly choose the order in which I test queens. This leads to wildly inconsistent behavior, where sometimes it performs amazingly and other times terribly. I suspect that if I added random restarts, it would work even better than my hill-climbing, even as it would be extremely stochastic.

This table was made in the same way as Hill Climbing, where I ran the algorithm 100 times and averaged the results. Given the variance of this algorithm, that was necessary to get accurate data.

n	number_of_iterations	number_of_moves	time (seconds)
10	67.39	124.78	0.0004276418686
11	91.52	172.04	0.0005479192734
12	110.26	208.52	0.0006678462029
13	165.03	317.06	0.001025519371
14	152.73	291.46	0.001060805321
15	207.54	400.08	0.001442234516
16	204.76	393.52	0.001338984966
17	234.21	451.42	0.001594479084
18	214.65	411.3	0.001557514668
19	360.25	701.5	0.002385950089
20	470.28	920.56	0.002983646393
21	292.55	564.1	0.001739442348
22	643.26	1264.52	0.004358673096
23	526.34	1029.68	0.00350612402

24	374.54	725.08	0.002406718731
25	410.31	795.62	0.002417955399
26	364.13	702.26	0.002125716209
27	992.7	1958.4	0.006734297276
28	1508.83	2989.66	0.01017973185
29	1113.46	2197.92	0.007373092175
30	2744.08	5458.16	0.01705914974
31	2937.99	5844.98	0.01764408827
32	1099.87	2167.74	0.007069106102
33	1324.86	2616.72	0.008119785786
34	1889.53	3745.06	0.0119366765
35	5341	10647	0.03295289993
36	2007.8	3979.6	0.01258448839
37	2424.34	4811.68	0.015233953
38	4236.41	8434.82	0.02812412024
39	2277.68	4516.36	0.01360116005
40	1377.43	2714.86	0.008267858028
50	14175.02	28300.04	0.08823185682
60	101659.2	203258.4	0.5967064357

Which algorithm is better changes depending on what the side-length is. At n=40, Hill Climbing takes 0.0972 seconds per iteration, while Forward Checking, on average, takes 0.00827 seconds per iteration, which is clearly superior. However, when we increase n to 60, Hill Climbing takes 0.357 seconds, while Forward Checking takes 0.597 seconds, which is inferior. And at n=70, Hill Climbing takes 0.489 seconds, while Forward Checking takes so long that I was unwilling to simulate it 100 times (when I simulated it 50 times, it took a bit over 3 seconds on average). So it looks like in the long run, my Hill-Climbing algorithm is superior to Forward Checking (and far more reliable, not that I measured the variance).

According to my numbers, Forward Checking always has much more iterations and moves at any level, but you should ignore that. It only has more iterations because of a quirk in how I measure data. You said on Piazza that any iteration that doesn't actually involve placing a queen, only ruling out where to place it, doesn't count as an iteration. Because of that, the many iterations within and applying my fitness function to determine which queen should be moved don't count as iterations, even though it takes $O(n^2)$ time to run. When I originally ran the algorithm counting all of those loops, I had a number of iterations in the millions by n=40.

Likewise, ignore that Forward Checking has more moves. It has much more moves because it is much more brute-force than my Hill-Climbing algorithm. Hill-Climbing takes $O(n^2)$ time determining where to place a queen, while Forward Checking only takes $O(n)$ time determining where not to put it. Because of this, Forward Checking makes more, dumber iterations, but takes less time doing so.