Lab Report A3
CS4300
Derek Heldt-Werle 2,4,6
Matthew Lemon 1,3,5
9/22/16

1. Introduction

Author: Matthew Lemon
In this lab we will be conducting a comparison of the AC-1 and AC-3 functions on the N-queens problem. We will be utilizing the algorithms laid out in the 1977 paper written by Alan Mackworth to implement the two algorithms. We intend to answer the following questions:

- What is the conditional expected reduction in ones (for each N separately) for a given starting number of ones?
- What is the relative execution time ratio of the two algorithms as a function of N?

2. Method

Author: Derek Heldt-Werle

In terms of implementation, we were heavily influenced by the 1977 paper by Alan Mackworth that lays out both AC-1 and AC-3.

For AC-1, like the paper describes, we take every arc and throw it into a queue. From here we will continue to delete arcs from the queue until we have iterated and no change has been made to the matrix in an entire pass of the algorithm.

For AC-3, we take a similar approach. In this algorithm we use a queue structure, but do not allow for duplicates to be added to the queue to be reexamined. To implement the queue for AC-3, in an effort to increase efficiency, we effectively created a doubly linked list that is preallocated to the potential max size in terms of the number of arcs we will visit.

We ran both AC-1 and AC-3 1200 times for each given size of N. Each successive 200 runs using each of the algorithms had a 20% greater chance of containing 0's, or locations in which the queen can not be placed. Timing data was captured through the use of the built in tic and toc functions.

3. Verification of Program

Author: Matthew Lemon
G = [0,1,1,1; 1,0,1,1; 1,1,0,1; 1,1,1,0]
D = [1,1,1,1; 1,0,0,0; 1,1,1,1; 1,1,1,1]
P = CS4300_P_no_attack
AC-1:
Results of running CS4300_AC1(G, D, P)
[0,0,1,0; 1,0,0,0; 0,0,0,1; 0,1,0,0]

AC-3:
Results of running CS4300_AC3(G, D, P)
[0,0,1,0; 1,0,0,0; 0,0,0,1; 0,1,0,0]

Computation of reduction by hand:

$$\begin{bmatrix} 1&1&1&1\\ 1&0&0&0\\ 1&1&1&1\\ 1&1&1&1 \end{bmatrix} \to \begin{bmatrix} 0&0&1&1\\ 1&0&0&0\\ 1&1&1&1\\ 1&1&1&1 \end{bmatrix} \to \begin{bmatrix} 0&0&1&1\\ 1&0&0&0\\ 0&0&1&1\\ 1&1&1&1 \end{bmatrix} \to \begin{bmatrix} 0&0&1&1\\ 1&0&0&0\\ 0&0&1&1\\ 0&1&0&1 \end{bmatrix}$$

$$\to \begin{bmatrix} 0&0&1&1\\ 1&0&0&0\\ 0&0&0&1\\ 0&1&0&1 \end{bmatrix} \to \begin{bmatrix} 0&0&1&0\\ 1&0&0&0\\ 0&0&0&1\\ 0&1&0&1 \end{bmatrix} \to \begin{bmatrix} 0&0&1&0\\ 1&0&0&0\\ 0&0&0&1\\ 0&1&0&0 \end{bmatrix}$$

The functions are able to compute the same reduction that I do by hand.

G = [0,1,1,1; 1,0,1,1; 1,1,0,1; 1,1,1,0]
D = [1,0,0,0; 1,1,1,1; 1,1,1,1; 1,1,1,1]
P = CS4300_P_no_attack
AC-1:
Results of running CS4300_AC1(G, D, P)
[0,0,0,0; 0,0,0,0; 0,0,0,0; 0,0,0,0]

AC-3:
Results of running CS4300_AC3(G, D, P)
[0,0,0,0; 0,0,0,0; 0,0,0,0; 0,0,0,0]

Computation of reduction by hand:

$$\begin{bmatrix}1&0&0&0\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix} \rightarrow \begin{bmatrix}1&0&0&0\\0&0&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix} \rightarrow \begin{bmatrix}1&0&0&0\\0&0&1&1\\0&1&0&1\\1&1&1&1\end{bmatrix} \rightarrow \begin{bmatrix}1&0&0&0\\0&0&1&1\\0&1&0&1\\0&1&1&0\end{bmatrix}$$

$$\rightarrow \begin{bmatrix}1&0&0&0\\0&0&0&1\\0&1&0&1\\0&1&1&0\end{bmatrix} \rightarrow \begin{bmatrix}1&0&0&0\\0&0&0&1\\0&1&0&0\\0&1&1&0\end{bmatrix} \rightarrow \begin{bmatrix}1&0&0&0\\0&0&0&1\\0&1&0&0\\0&0&0&0\end{bmatrix} \rightarrow \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

4. Data and Analysis

Author Derek Heldt-Werle

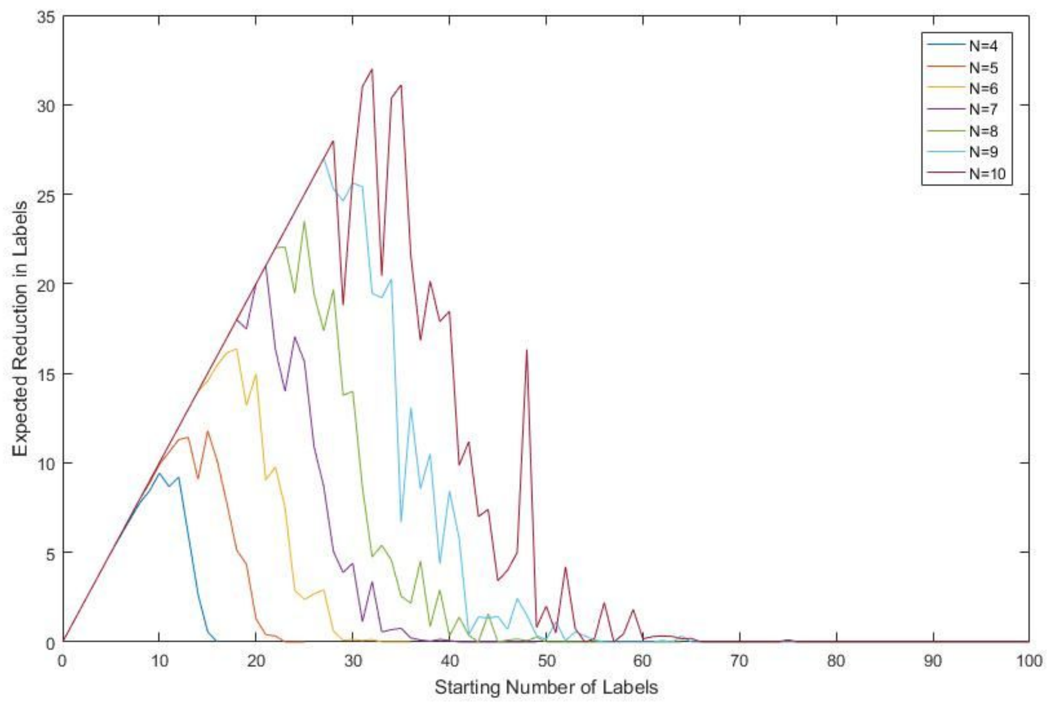

Figure 1: Shows for each N, the experiments with N starting ones and average the reduction found for those experiments.

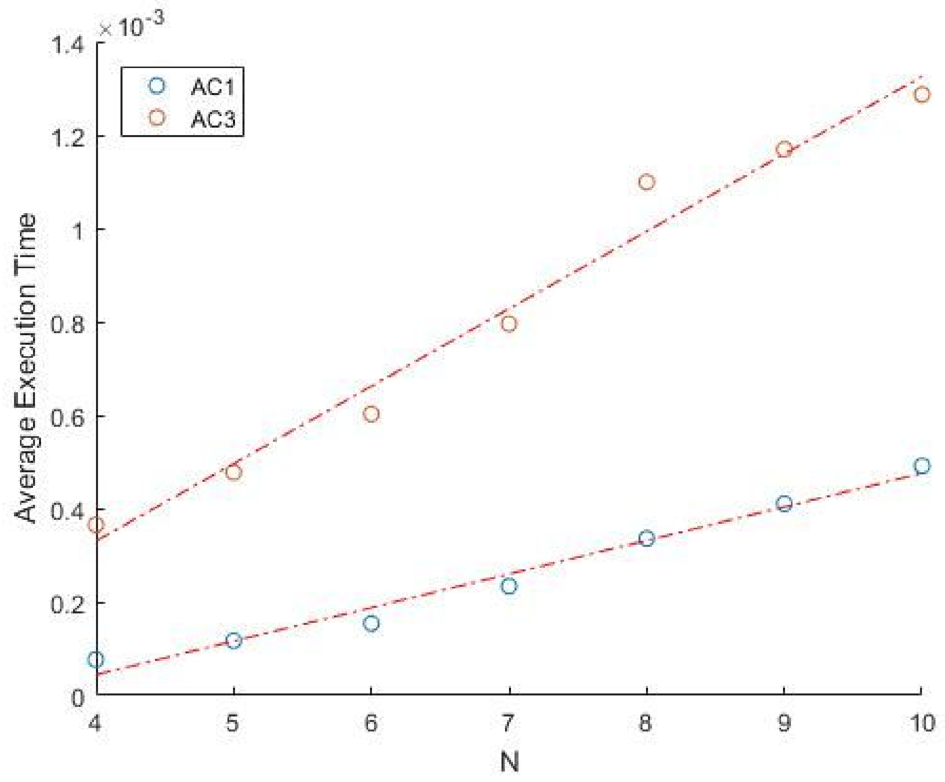

Figure 2: Shows the average execution time for each size N

The above graph shows that our implementation of AC-1 runs faster than our AC-3, on average, for all N size inputs. The only time in which we saw the two algorithms perform at equal speeds were when the entire boards were filled with ones, as expected.

P	N	Ones	r1	r2	t1	t2
0	4	0	0	0	2.77E-05	4.65E-05
0.2	4	3.235	3.235	3.235	3.92E-05	9.22E-05
0.4	4	6.31	6.13	6.13	4.36E-05	9.68E-05
0.6	4	9.7	7.855	7.855	5.78E-05	0.000134
0.8	4	12.605	5.25	5.25	6.04E-05	0.00011
1	4	16	0	0	6.52E-05	3.40E-05
0	5	0	0	0	5.03E-05	4.23E-05
0.2	5	4.91	4.885	4.885	6.72E-05	0.000183
0.4	5	9.835	9.325	9.325	7.84E-05	0.00029
0.6	5	15.15	9.065	9.065	9.81E-05	0.000318

0.8	5	20.005	2.575	2.575	7.94E-05	0.000133
1	5	25	0	0	5.76E-05	5.21E-05
0	6	0	0	0	5.52E-05	4.74E-05
0.2	6	7.57	7.57	7.57	9.59E-05	0.000426
0.4	6	14.3	13.51	13.51	0.000127	0.000742
0.6	6	21.32	9.795	9.795	0.000155	0.000646
0.8	6	28.695	0.77	0.77	0.000106	0.00013
1	6	36	0	0	8.85E-05	8.15E-05
0	7	0	0	0	7.71E-05	6.63E-05
0.2	7	9.89	9.89	9.89	0.000155	0.000968
0.4	7	19.595	17.105	17.105	0.0002	0.001629
0.6	7	29.12	5.575	5.575	0.000201	0.000687
0.8	7	38.795	0.46	0.46	0.00014	0.000163
1	7	49	0	0	0.000135	0.000126
0	8	0	0	0	0.000102	8.75E-05
0.2	8	12.87	12.87	12.87	0.000187	0.001655
0.4	8	25.955	18.005	18.005	0.000286	0.0026
0.6	8	38.62	2.26	2.26	0.000238	0.000538
0.8	8	50.915	0.05	0.05	0.000189	0.000176
1	8	64	0	0	0.000193	0.000179
0	9	0	0	0	0.000129	0.000111
0.2	9	16.345	16.345	16.345	0.000251	0.00292
0.4	9	32.58	17.415	17.415	0.000428	0.004205
0.6	9	48.845	0.675	0.675	0.000278	0.000387
0.8	9	64.555	0.015	0.015	0.000263	0.000235
1	9	81	0	0	0.000271	0.000253
0	10	0	0	0	0.000166	0.000143
0.2	10	20.2	20.04	20.04	0.000331	0.004813
0.4	10	39.63	16.195	16.195	0.000504	0.00506
0.6	10	59.85	0.375	0.375	0.000397	0.00049
0.8	10	80.315	0	0	0.00038	0.000349

1	10	100	0	0	0.000373	0.000348

Figure 3: The data above describes the graph found in Figure 1.

5. Interpretation

Author: Matthew Lemon
The graph in Figure 1 shows the expected reduction per number of ones for each N.
For each N the reduction of ones is linear until the number of ones > N after which point the reduction drops as the number of ones gets closer to N*N. This is expected as there needs to be at least one one in each row/column for there to be a solution so if there are less the graph would go to the zero matrix. Then as the number of ones approaches N*N we get closer to a complete graph filled with ones which would have zero reduction.

After we implemented AC-1 and AC-3 we ran our timing code to compare them and found that an average AC-1 would run faster than AC-3, we assumed we had implemented something incorrectly and went back to revise our code and optimized and fixed as much as we could and ended up with AC-1 still being faster than AC-3. Inspecting the data from the trials we ran (Figure 3) we found that AC-3 would slow down considerably when p (the probability of a 1) was 0.2 and 0.4. Otherwise AC-1 and AC-3 were actually quite similar and AC-3 could even be faster for $p > 0.4$ and $p < 0.2$.

6. Critique

Author Derek Heldt-Werle

Given that we had access to the pseudo-code for both of the algorithms, we were able to leverage this to our advantage when creating our codes. During this assignment we learned a great deal about arc-consistency through both online resources and the 1977 paper written by Mackworth. We also learned a lot about working with the built in graphing functions. In order to properly replicate the necessary graphs we had to spend a great deal of time develop an understanding of linear regression, and how to properly implement it with the data we generated.

For this lab we ran into a couple of issues that increased the time we spent on this significantly. The main issue we encountered during this assignment was determining the reasoning behind why our AC-3 ran slower than our AC-1. In an effort to combat this, we made various attempts to increase the speed at which AC-3 ran through various memory allocation optimizations as well as the creation of a simple linked list structure. Despite this, we were unable to bring the speed of our AC-3 up to speed with our AC-1. The other issue we ran into was getting the graphs to turn out properly. Since our only exposure to graphing was quite limited, reproducing the graphs took quite some time.

For the future we will continue to grow in our abilities in matlab, as it is becoming increasingly easy to develop in the language.

7. Log

Author Matthew Lemon & Derek Heldt-Werle
Coding Portion (Worked together): 10
Report (Derek): 3
Report (Matt): 3