Assignment 3 – CS4300

Arc Consistency Algorithms

Braden Scothern & Kyle Heaton

# Introduction

In order to explore Arc Consistency, we have measured the AC-1 and AC-3 algorithms when applied to the *N*-queens problem using a series of tests. The tests cover the range of *N* = 4:10 with 200 randomly generated *N* x *N* boards for each percentage *p*, of ones which varies from 0 to 1 in steps of 0.2. This results in 1200 boards being tested for each *N*. As these tests are run we have answered the following questions

* What is the number of ones before and after the application of the constraint algorithms?
* What is the execution time of each algorithm for each trial (using tic and toc)?

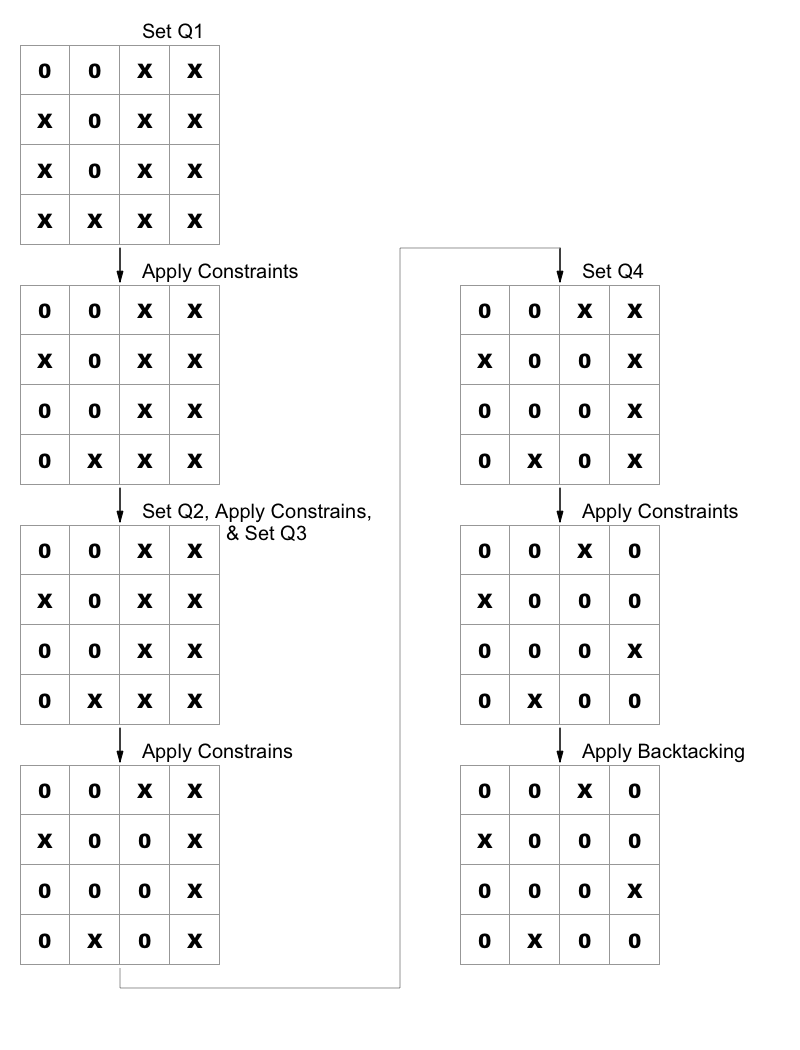
# Method

Our method on this assignment was wholly unsuccessful. The plan however was to first, randomly generate a domain matrix according to all of the values of n and p. Second track how many ones were in that domain before and after running the Arc Consistency functions on them. Since the results of Ac1 and Ac3 should produce the same reductions, only Ac3 was recorded and used for reduction calculations. Third, time how long it would take AC1 and AC3 to run and experimentally determine then compare their complexities.

Despite the extension given to us, we were unable to achieve success on this assignment. This was caused by poor planning, stress caused by extenuating circumstances in personal lives, as well as just struggling with core concepts of the algorithm. We are excited to see the solution to behold what evaded us for so many hours

# Verification of Program

In order to make sure that the logic of checking that a queen has been placed in the correct location we have calculated the following example by hand. The application of constrains looks at all arcs and makes sure there is support from some other element in that queen’s domain, main of these tiny incremental steps have their details omitted because they would take too much space to represent. We have also assumed that input will match the form of our verification, that is that it looks identical from the right side up perspective. In other words, Queens are placed by column not by row. We have made these assumptions based on 2 things. First the expected input matches the examples we examined in class. The second is the output of eye() because it was used in the example pseudocode from class, in our own testing, and because it was far more convenient to define it this way.



Results from Matlab.

>> G = ~eye(4,4);

>> D = [0,0,1,1; 1,0,1,1; 1,0,1,1; 1,1,1,1];

>> CS4300\_AC3(G, D, ‘CS4300\_P\_no\_attack’);

>> ans =

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

# Data and Analysis

I would love to put some information in here, but I wasn’t ever able to get the data into some manageable form. I believe that our data handling and plotting was mostly correct, it just took a ridiculous amount of time to run. Being unfamiliar with tricks of Matlab, I was unable to substantially shorten runtime. Testing with smaller numbers of trials, did yield results, but nothing like expected

# Interpretation

Looking at the questions posed in section one we have the following results from our testing.

* The average number of ones before and after the application of the constraint algorithms is:

We were unable to answer this question

* The average execution time of each algorithm for each trial (using tic and toc) is:
  + AC1
    - 0.0690
    - 0.0480
    - 0.0805
    - 0.1453
    - 0.2260
    - 0.3617
    - 0.5384
  + AC3
    - 0.0690
    - 0.0411
    - 0.0713
    - 0.1274
    - 0.2039
    - 0.3320
    - 0.5097

# Critique

The hardest part of this assignment for me was Matlab. I can’t blame my lack of progress on the language, but that is where I struggled the most. When trying to do averages of reduction results, I had so many for loops that I got half way through and had to start over because it was such a mess I didn’t know what was going on. I am sure there is a Matlab way of doing a lot of the things, but they evaded me. I really would have liked to a better explanation on what kind of data was expected. I liked the graphs, but I had questions that I didn’t ask soon enough. This assignment for me was a difficult balance between trying to figure out things on my own, and feeling like I am asking for a handout from the professor.

# Log

Braden Scothern

* 14 hours – Writing AC1, AC3, and CS4300\_P\_no\_attack code
* 3.5 hours – Formatting and writing report sections 1, 3, & 5

Kyle Heaton

* 5 hours getting framework surrounding AC1 and AC3 to be able to generate domain matrices and pull the data we need from tests
* 8 hours trying to get the data that we are pulling from AC1 and AC3 into some manageable form to plot it and actually be able to glean some knowledge from the graphs
* 1 hour writing sections 2, 4, 6 in lab report
* 5 hours trying to wrap everything up

# Appendix

MATLAB Code Files with Brief description:

* CS4300\_AC1(G,D,P) -- This implements the AC1 algorithm for the N-queens problem.
* CS4300\_AC3(G,D,P) – This implements the AC3 algorithm for the N-queens problem.
* CS4300\_Arc\_Consistency(G,D,P) – This is a helper function which is used to run both AC1 and AC3 algorithms and calculate execution times for evaluation.
* CS4300\_Count\_Ones – This will take a (M x N) array and see how many elements are 1 and return that value.
* CS4300\_Generate\_D(size, percentage) – This creates an array of (size x size) with each element having percentage (with a range of 0 – 1.0) of being 1.
* CS4300\_P\_no\_attack(I,a,j,b) – This is used to check 2 locations in a matrix to see if they can possibly attack one another.
* CS4300\_Run\_Tests() – This runs all of the main loops and does the main calculations for our assignment.