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

A lot of the work for the A\* algorithm is split out into helper functions; most of the body of the actual function is verification code and the manipulation of arrays. The first helper function that is used is CS4300\_A2\_Expand\_States(). It takes a current state and returns a 3x3 array of integers with the first row being the forward action, the second being the right turn, and the third being the left turn. We then loop over each row and validate that the state is in bounds, not a duplicate and that it not a death state. We verify that a state isn’t a duplicate with the function CS4300\_State\_Is\_Duplicate() which takes the entire tree of nodes and makes sure that the potentially new node cannot be found in it.

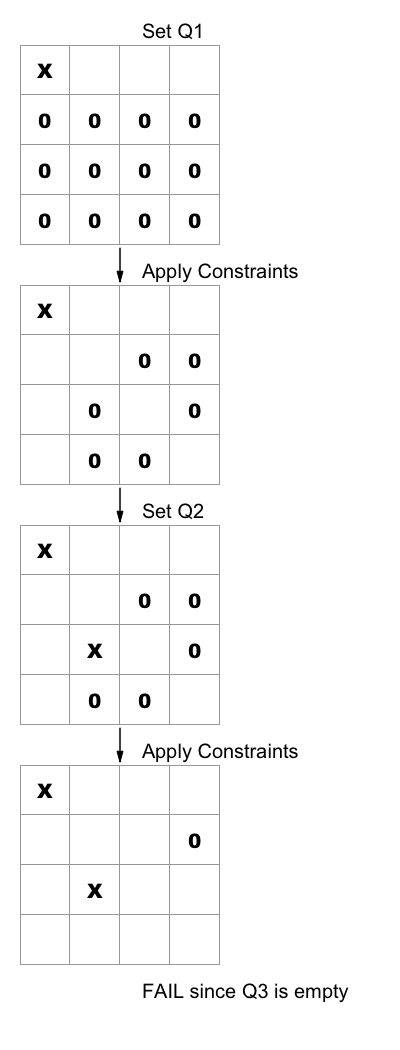
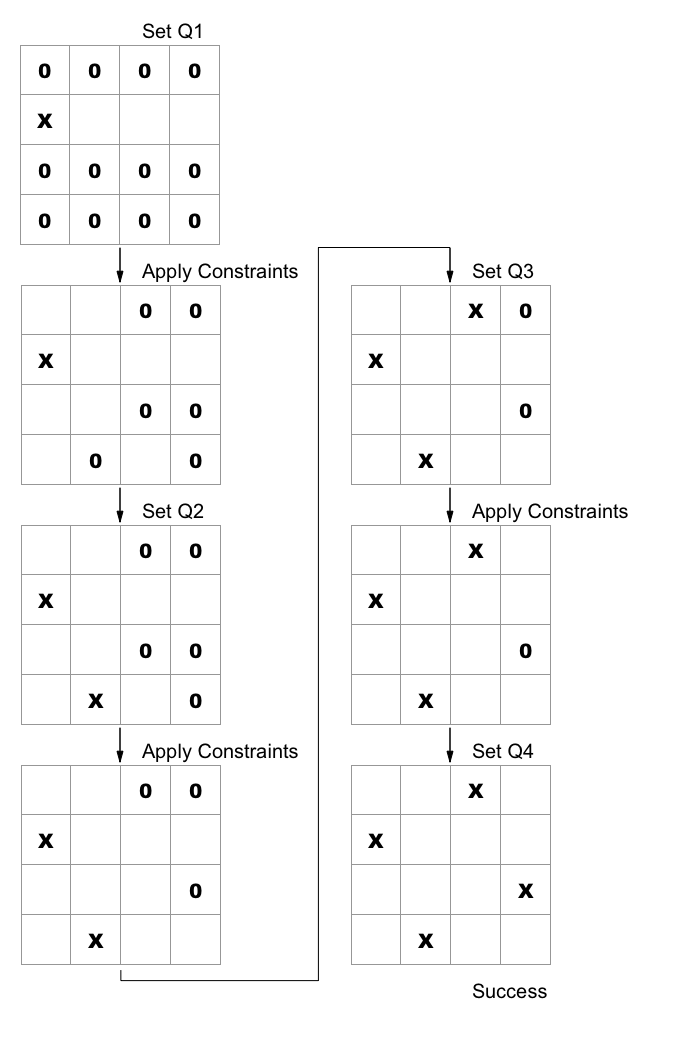
When a node is pulled from the frontier it is immediately checked for a goal state. If it is a valid solution that solution is returned. Otherwise the node is expanded and all valid children are immediately put on the tree. If the state is valid but not the goal then we will add it as a new node and make it a child of the main node we are currently looking at. Once we have all of these new children we add them to the frontier according to the logic needed to satisfy the option code given to the A\* function. All of this logic is looped over as long as the current node hasn’t reached a goal state and as long as we have more nodes in the frontier for us to search.

When it isn’t possible to find a path to the gold, our A\* function will return an empty solution path and a set of nodes that cover all reachable locations on the board.

The method used here is simply to generate a large number of samples and compute the mean, variance and confidence of the result. An alternative would be to run a large number of trials where each trial would get a fixed number of samples from rand, then compute the mean and variance of each trial, and then compute the mean and variance over all those trials. This latter approach was not implemented.

# Verification of Program

In order to make sure that the logic of checking that a queen has been placed in the correct location we will calculate some examples by hand.



**Example 2**

**Example 1**

Results from Matlab.

* Example 1
* Example 2

# 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 found myself having lots and lots of matrices that were mostly 0s, and a little bit of data in them. I spent most of my time on this assignment setting up a structure to be able to easily extract results from our tests and present them, but I failed miserably.

# Interpretation

# 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

* 4 hours – Writing and debugging code
* 3 hours – Writing report

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

# Appendix

MATLAB Code Files with Brief description: