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.

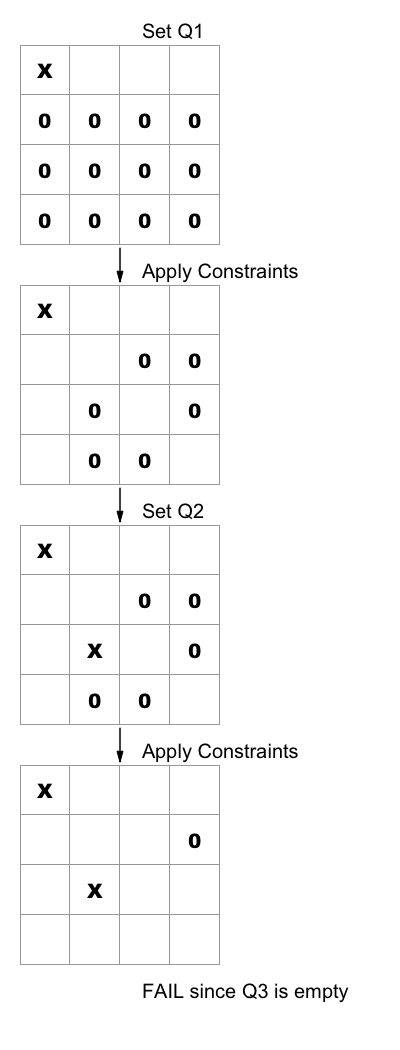
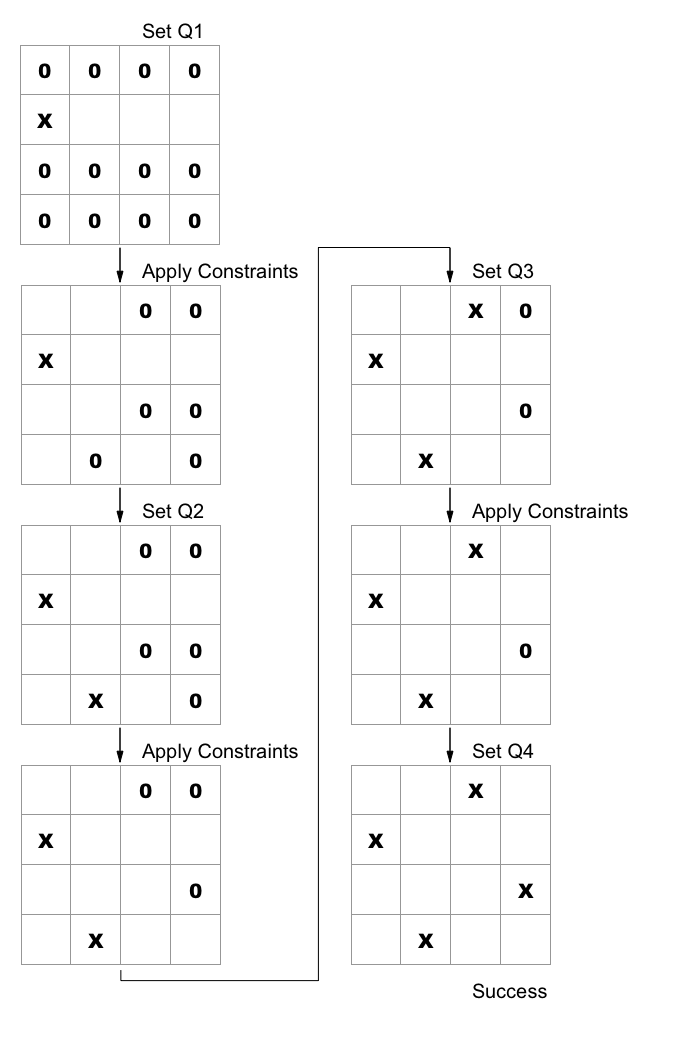
When a node is pulled from the frontier it is immediately checked for a goal state. If it is a valid solution that solution is returned. Otherwise the node is expanded and all valid children are immediately put on the tree. If the state is valid but not the goal then we will add it as a new node and 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.

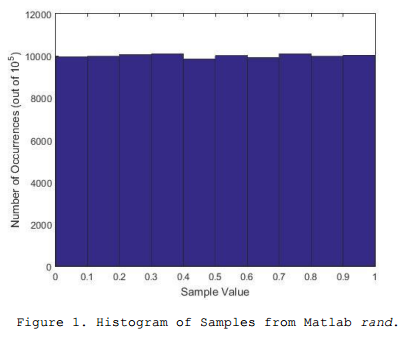
* Example 1
* Example 2

# Data and Analysis

Description of data and how it is used/organized for training and testing  
Results from experiments (graphs, tables, means, variance, confidence intervals)  
Describe relationships found or qualitative description of results  
Discuss implications of statistical results

Give 4x4 tables with the number of nodes in the search tree for each of these options when the gold is at (x.y)  
Plot actual size of individual trial results and a histogram

Figure 1 shows the data collected from Matlab rand.



# Interpretation

ANSWER QUESTIONS

Answer questions posed in into, use analysis to support conclusions  
Discuss future work and extensions

The results produced by rand are very close to the theoretical values of the mean and variance and the confidence interval at the 95% level is very short. Thus, the mean and variance results are close enough to the theoretical values to be acceptable in most applications.

# Critique

This experiment is very effective at introducing a high level of complexity that comes with finding optimal solutions for even very simple problems. The concepts were all fairly straightforward and easy to understand as they were worked out by hand and in the process of writing code. The biggest suggestion to improve this experiment is to standardize the proper definition of the heuristic function in A\* so multiple functions can be used without the potential of needing to modify the function call inside of the A\* function.

# Log

Braden Scothern

* 240 min – Writing and debugging code
* 120 min – Writing report

Kyle Heaton

* 57 min – Reviewing assignment, fleshing out requirements, outlining report
* 5 hours – Writing helper functions and writing report sections 1, 3, and 5

# Appendix

MATLAB Code Files with Brief description:

* + CS4300\_A2\_20percent\_Pit\_Board.m – Helper function to generate Wumpus World boards. The gold and wumpus are placed randomly, then every empty cell has a 20% change of having a pit.
  + CS4300\_ A2\_Calculate\_Tree\_Size.m – Helper function to determine the size of the tree for tree node statistics
  + CS4300\_ A2\_Expanded\_States.m – Helper function to generate all of the possible reachable states from a state that has been passed in
  + CS4300\_ A2\_Manhattan\_Distance.m – Helper function to calculate the Manhattan distance for a state from the starting point
  + CS4300\_ A2\_Wumpus\_A\_star1.m – An A\* search algorithm to search a given wumpus board for a path to the gold