CP468 – A2

Pillip Lee

Derek Cruise

Scott Cosentino

Aman Kumar

Group 11

2017-11-03

Table of Contents

Problem Statement

The purpose of this assignment is to implement AC-3 algorithms to enforce arc-consistency to an arbitrary 9 x 9 Sudoku puzzle(s). An execution of the algorithm should report whether the equivalent arc-consistent CSP is found. If the puzzle is solved, a solution is reported. If the puzzle is not solved, an additional algorithm is implemented to solve the puzzle in its entirety and a solution is reported. In addition, throughout the implementation, the code must also keep track of the length of the queue at each step of the AC-3 algorithm.

CSP Representation (Array and Constraints)

The Variable array holds the variables, they are stored as x, y pairs that correspond to the location of the square on the Sudoku board, they range from (0,0) to (8,8).

The Domain array holds the domain for each variable, the domain at position “I” corresponds with the variable at position “I” in the variable array.

The Constraint array holds pairs of variables, each variable is stored as it’s index in the variable array. Every constraint is such that the variable on the left cannot equal the variable on the right.

Flattening the alldiff constraints into binary constraints and removing duplicates resulted in 1,620 total constraints.

Implementation

```
#Sudoku Solver
#to solve as a CSP it uses AC-3 to preprocess sudoku then uses Human logic to solve
the sudoku then uses backtracking if that fails
#to solve as a 2d-array it uses human logic then backtracking if that fails

from __future__ import print_function
import time
import copy
import math
import numpy as np
from matplotlib import pyplot as plt

Start = [] #Given Matrix
Solved = [] #Solution Matrix

Queue_Length = []

#V = [[0,0],[1,0]]
#D = [[1,2,3,4,5,6,7,8,9],[1,2]]
#C = [[0,1],[1,0]] Does not Equal
#C contains variable indexes in the V list

def AC_Three(V,D,C):
    Q = []
    Q = copy.deepcopy(C)
    return AC_Three_Given_Queue(V,D,C,Q,True)
```

```
#performs AC_Three with a given queue
def AC_Three_Given_Queue(V,D,C,Q,report):
    valid = True
    global Queue_Length
    while valid and len(Q) != 0:
        if len(D[Q[0][1]]) == 1:
            if D[Q[0][1]][0] in D[Q[0][0]]:
                D[Q[0][0]].remove(D[Q[0][1]][0])
                for i in C:
                    if i[1] == Q[0][0]:
                        if i not in Q:
                            Q.append(copy.deepcopy(i))
        elif len(D[Q[0][1]]) == 0:
            valid = False
        del Q[0]

        if report:
            Queue_Length.append(len(Q))
    return valid

def visualize_Queue(queue):
    x, y = range(0,len(queue)), queue
    plt.scatter(x, y, alpha=0.9)
    plt.title('Sudoku Solver using AC-3 by Group11')
    plt.xlabel('Step #')
    plt.ylabel('Length of the Queue')
    axes = plt.gca()
    axes.set_xlim(xmin=0)
    axes.set_ylim(ymin=0)
    plt.show()
    return None

#attempts to solve the sudoku the same way the average person does as a CSP
def human_logic_CSP(V,D,C):
    moves = 1
    while moves > 0: #loops until no moves are made
        moves = 0
        changed = []

        rowhas = [[[0,0] for k in range(0,9)] for j in range(0,9)] #a counter of how
many places in a row can have each number
        colhas = [[[0,0] for k in range(0,9)] for j in range(0,9)] #a counter of how
many places in a column can have each number
        boxhas = [[[0,0] for k in range(0,9)] for j in range(0,9)] #a counter of how
many places in a box can have each number

        for i in range(0,len(V)): #fill the lists
            if len(D[i]) != 1:
                for x in D[i]:
                    rowhas[V[i][0]][x-1][0] += 1
                    rowhas[V[i][0]][x-1][1] = i
                    colhas[V[i][1]][x-1][0] +=1
                    colhas[V[i][1]][x-1][1] = i
                    boxhas[V[i][0]//3*3+V[i][1]//3][x-1][0] += 1
                    boxhas[V[i][0]//3*3+V[i][1]//3][x-1][1] = i
```

```
        for x in range(0,9): #check if only one place can have the value
            for y in range(0,9):
                if rowhas[x][y][0] == 1:
                    D[rowhas[x][y][1]] = [y+1]
                    changed.append(rowhas[x][y][1])
                if colhas[x][y][0] == 1:
                    D[colhas[x][y][1]] = [y+1]
                    changed.append(colhas[x][y][1])
                if boxhas[x][y][0] == 1:
                    D[boxhas[x][y][1]] = [y+1]
                    changed.append(boxhas[x][y][1])

        Q = []
        for u in C:
            if u[1] in changed:
                Q.append(copy.deepcopy(u))
        AC_Three_Given_Queue(V,D,C,Q,False)
        moves = len(changed)

#finds the first blank space and trys all possible moves
def rec_trymove_CSP(V,D,C):
    global Solved
    if len(Solved) == 0: #if it's not already solved

        min = 10
        minplace = -1
        #loop to find first blank space
        for i in range(0,len(V)):
            if len(D[i]) != 1 and len(D[i]) < min:
                min = len(D[i])
                minplace = i

        if min > 1 and min < 10:
            i = minplace
            for move in D[i]:
                D2 = copy.deepcopy(D)
                D2[i] = [move]
                Q = []
                for u in C:
                    if u[1] == i:
                        Q.append(copy.deepcopy(u))
                if AC_Three_Given_Queue(V,D2,C,Q,False):
                    rec_trymove_CSP(V,D2,C)

        else:
            global Start
            Solved = copy.deepcopy(Start)
            for i in range(0,len(V)): #make any moves that AC-3 determined
                if len(D[i]) == 1:
                    Solved[V[i][0]][V[i][1]] = D[i][0]

#Solves the given sudoku as a CSP
def solve_sudoku_CSP(Start):
    #start_time = time.time()
```

```
    global Solved
    print("Given:")
    for x in range(0,9): #print the given matrix
        for y in range(0,9):
            if Start[x][y] == 0:
                print("-", end = " "),
            else:
                print(Start[x][y],end=" ")
        print("")
    print("")
    board = copy.deepcopy(Start)


    V = [] #Variable array
    D = [] #Domain array
    C = [] #Constraignt array
    for x in range(0,9): #populate V and D
        for y in range(0,9):
            V.append([x,y])
            if board[x][y] == 0:
                D.append([1,2,3,4,5,6,7,8,9])
            else:
                D.append([board[x][y]])

    for i in range(0,len(V)): #populate C
        for u in range(0,len(V)):
            if i != u and (V[i][0] == V[u][0] or V[i][1] == V[u][1] or (V[i][0]//3 ==
V[u][0]//3 and V[i][1]//3 == V[u][1]//3)):
                C.append([i,u])

    #Run AC-3 to pre-process array, sometimes will solve array
    valid = AC_Three(V,D,C)

    solved_by_AC_Three = True
    for x in D:
        if len(x) != 1:
            solved_by_AC_Three = False

    if solved_by_AC_Three:
        print("Solved by AC-3")
        Solved = copy.deepcopy(Start)
        for i in range(0,len(V)): #make any moves that AC-3 determined
            if len(D[i]) == 1:
                Solved[V[i][0]][V[i][1]] = D[i][0]
    elif valid:
        print("Not Solved by AC-3")
        #will apply basic sudoku solving logic. This step is to remove any low
hanging fruit to avoid or speed up recursion.
        human_logic_CSP(V,D,C)

        #if already solved, then it will simply save the solution to Solved. else it
will solve the sudoku recursivly
        rec_trymove_CSP(V,D,C)
```

```
    #print results
    if not valid:
        print("CSP determined invalid by AC-3")
    elif len(Solved) == 0:
        print("Unsolveable")
    else:
        print("Solution:")
        for x in range(0,9): #print the completed matrix
            for y in range(0,9):
                print(Solved[x][y],end=" ")
            print("")
    print("")
    #print("Took %.4f seconds to run" % (time.time() - start_time))

def readInputFile(fname):
    #sets arrays to hold value read from file, the final set of values, and one array
to format string to integers
    holder = []
    fileValues = []
    formattedints = []

    with open(fname) as f:
        #Reads a line, and splits it based on the comma
        line = f.readline()
        holder = line.strip().split(",")

        #For each of the values in the holder array, append the integer conversion to
the integer array
        for c in holder:
            if c != "":
                formattedints.append(int(c))

        while line:

            #Append the integer array to the return set
            fileValues.append(formattedints)

            #Reset the integer array and read a new line
            formattedints = []
            line = f.readline()
            holder = line.strip().split(",")
            for c in holder:
                if c != "":
                    formattedints.append(int(c))
    return fileValues

#Input is assumed to be in the same directory as the program
readList = readInputFile("Sudoku_Input.txt")
valuesRead = 0

for values in readList:
    #Appends the values to the main array
     Start.append(values)
     valuesRead = valuesRead + 1
```

```
    #Once it reaches the 9th value, solves the sodoku and resets the variables
     if valuesRead == 9:
         solve_sudoku_CSP(Start)
         visualize_Queue(Queue_Length)
         print("The number of steps taken by AC-3: " + str(len(Queue_Length)))
         Start = []
         Solved = []
         Queue_Length = []
         valuesRead = 0
         print("-----------------------------------")
```

Sudoku Input

Test	Input	Test	Input
1	0,8,0,0,0,0,2,0,0		0,0,1,0,2,0,0,0,0
	0,0,0,0,8,4,0,9,0		0,0,0,5,0,7,0,0,0
	0,0,6,3,2,0,0,1,0		0,0,4,0,0,0,1,0,0
	0,9,7,0,0,0,0,8,0		0,9,0,0,0,0,0,0,0
	8,0,0,9,0,3,0,0,2		5,0,0,0,0,0,0,7,3
	0,1,0,0,0,0,9,5,0		0,0,2,0,1,0,0,0,0
	0,7,0,0,4,5,8,0,0		0,0,0,0,4,0,0,0,9
	0,3,0,7,1,0,0,0,0		
	0,0,8,0,0,0,0,4,0		
2	0,0,3,0,2,0,6,0,0		
	9,0,0,3,0,5,0,0,1		
	0,0,1,8,0,6,4,0,0		
	0,0,8,1,0,2,9,0,0		
	7,0,0,0,0,0,0,0,8		
	0,0,6,7,0,8,2,0,0		
	0,0,2,6,0,9,5,0,0		
	8,0,0,2,0,3,0,0,9		
	0,0,5,0,1,0,3,0,0		
3	0,0,0,1,0,0,7,0,2		
	0,3,0,9,5,0,0,0,0		
	0,0,1,0,0,2,0,0,3		
	5,9,0,0,0,0,3,0,1		
	0,2,0,0,0,0,0,7,0		
	7,0,3,0,0,0,0,9,8		
	8,0,0,2,0,0,1,0,0		
	0,0,0,0,8,5,0,6,0		
	6,0,5,0,0,9,0,0,0		
4	8,0,0,0,0,0,0,0,0		
	0,0,3,6,0,0,0,0,0		
	0,7,0,0,9,0,2,0,0		
	0,5,0,0,0,7,0,0,0		
	0,0,0,0,4,5,7,0,0		
	0,0,0,1,0,0,0,3,0		
	0,0,1,0,0,0,0,6,8		
	0,0,8,5,0,0,0,1,0		
	0,9,0,0,0,0,4,0,0		
5	0,0,0,0,0,0,0,0,0		
	0,0,0,0,0,3,0,8,5		

Results

Scenario 1

python sudoku_final.py

Given:

- 8 - - - - 2 - -

- - - - 8 4 - 9 -

- - 6 3 2 - - 1 -

- 9 7 - - - - 8 -

8 - - 9 - 3 - - 2

- 1 - - - - 9 5 -

- 7 - - 4 5 8 - -

- 3 - 7 1 - - - -

- - 8 - - - - 4 -

Not Solved by AC-3

Solution:

7 8 4 1 9 6 2 3 5

3 2 1 5 8 4 6 9 7

9 5 6 3 2 7 4 1 8

2 9 7 4 5 1 3 8 6

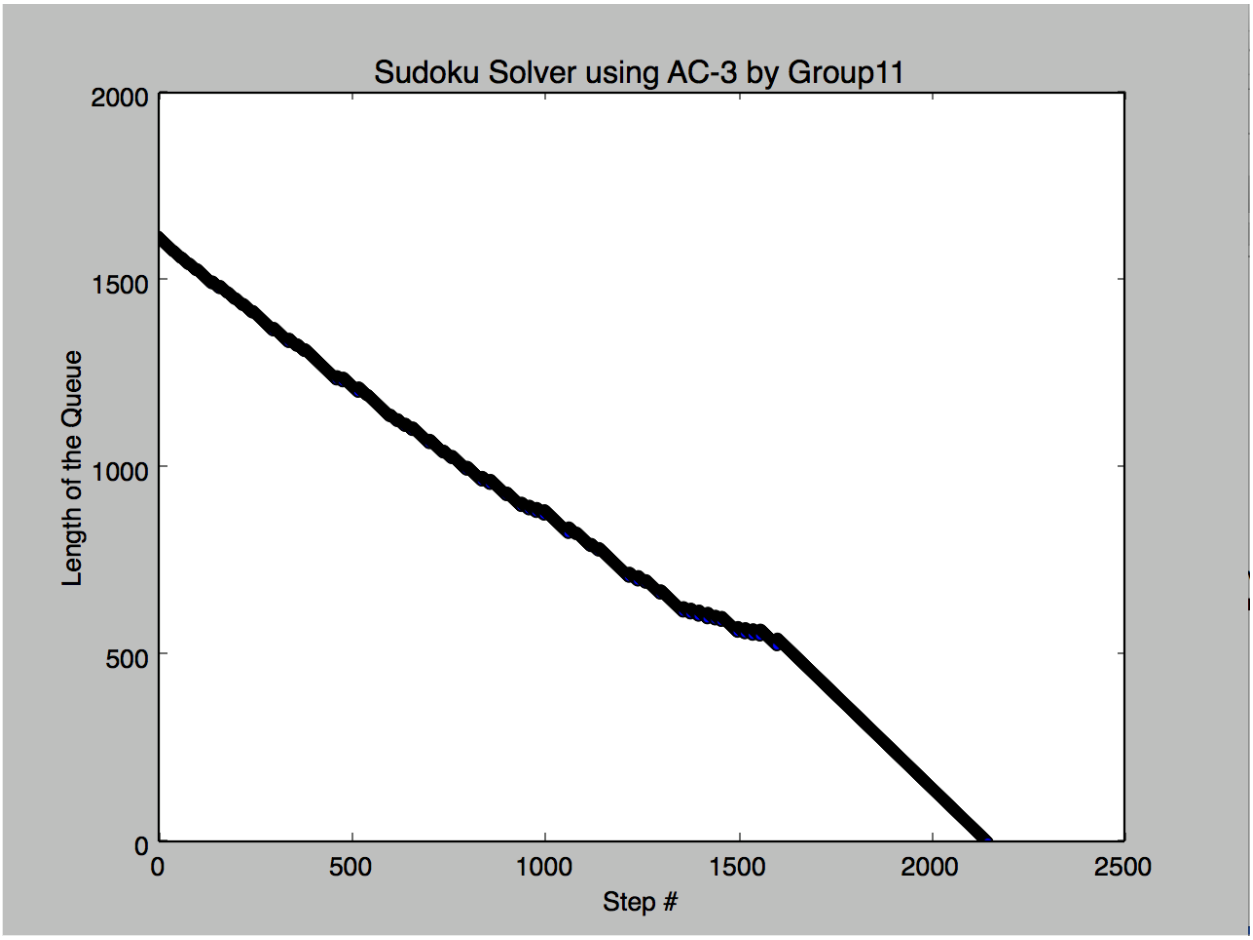
8 4 5 9 6 3 1 7 2

6 1 3 8 7 2 9 5 4

1 7 9 6 4 5 8 2 3

4 3 2 7 1 8 5 6 9

5 6 8 2 3 9 7 4 1



The number of steps taken by AC-3: 2150

Scenario 2

Given:

- - 3 - 2 - 6 - -

9 - - 3 - 5 - - 1

- - 1 8 - 6 4 - -

- - 8 1 - 2 9 - -

7 - - - - - - - 8

- - 6 7 - 8 2 - -

- - 2 6 - 9 5 - -

8 - - 2 - 3 - - 9

- - 5 - 1 - 3 - -

Solved by AC-3

Solution:

4 8 3 9 2 1 6 5 7

9 6 7 3 4 5 8 2 1

2 5 1 8 7 6 4 9 3

5 4 8 1 3 2 9 7 6

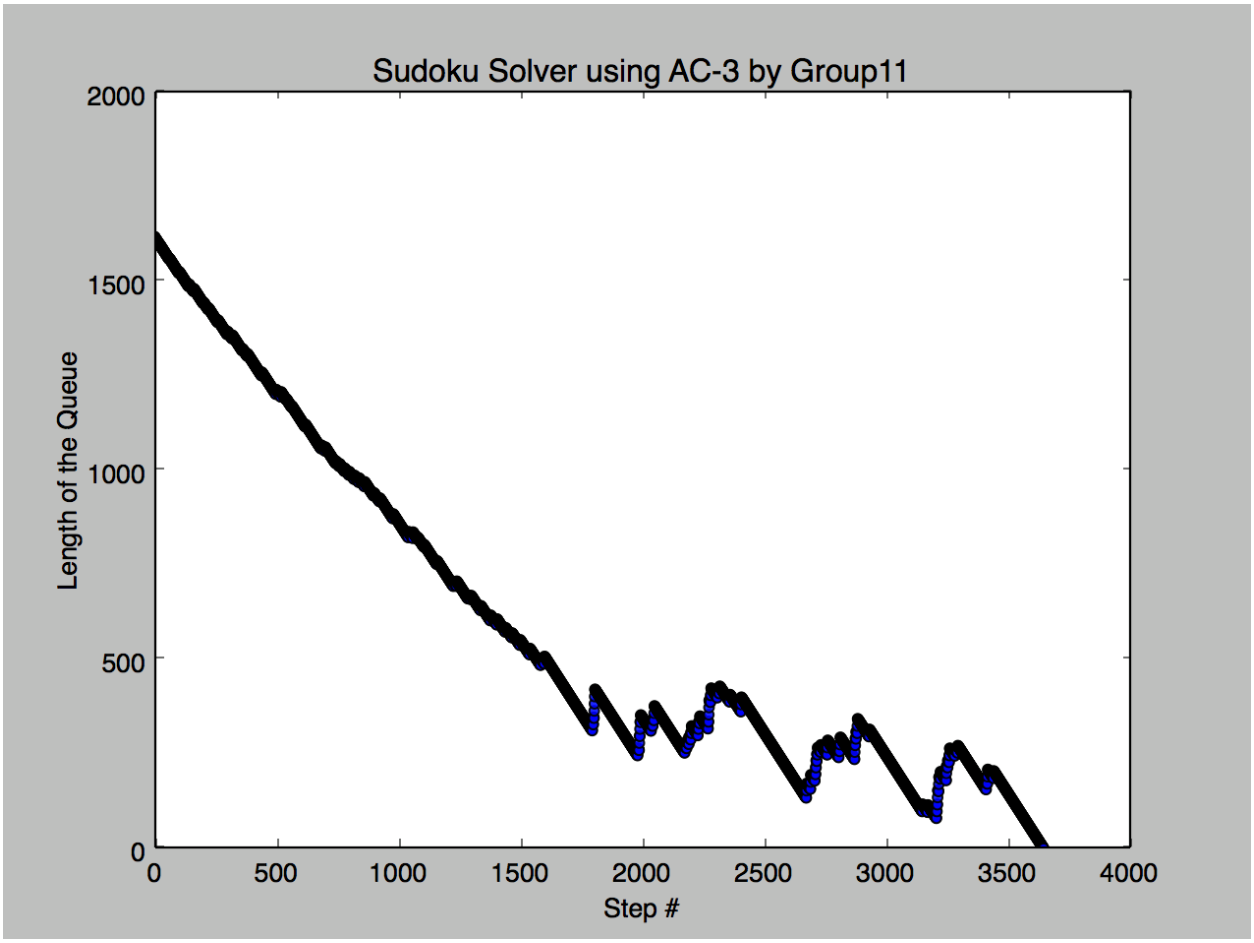
7 2 9 5 6 4 1 3 8

1 3 6 7 9 8 2 4 5

3 7 2 6 8 9 5 1 4

8 1 4 2 5 3 7 6 9

6 9 5 4 1 7 3 8 2



The number of steps taken by AC-3: 3650

Scenario 3

Given:

- - - 1 - - 7 - 2

- 3 - 9 5 - - - -

- - 1 - - 2 - - 3

5 9 - - - - 3 - 1

- 2 - - - - - 7 -

7 - 3 - - - - 9 8

8 - - 2 - - 1 - -

- - - - 8 5 - 6 -

6 - 5 - - 9 - - -

Not Solved by AC-3

Solution:

9 5 6 1 3 8 7 4 2

2 3 7 9 5 4 8 1 6

4 8 1 6 7 2 9 5 3

5 9 4 8 6 7 3 2 1

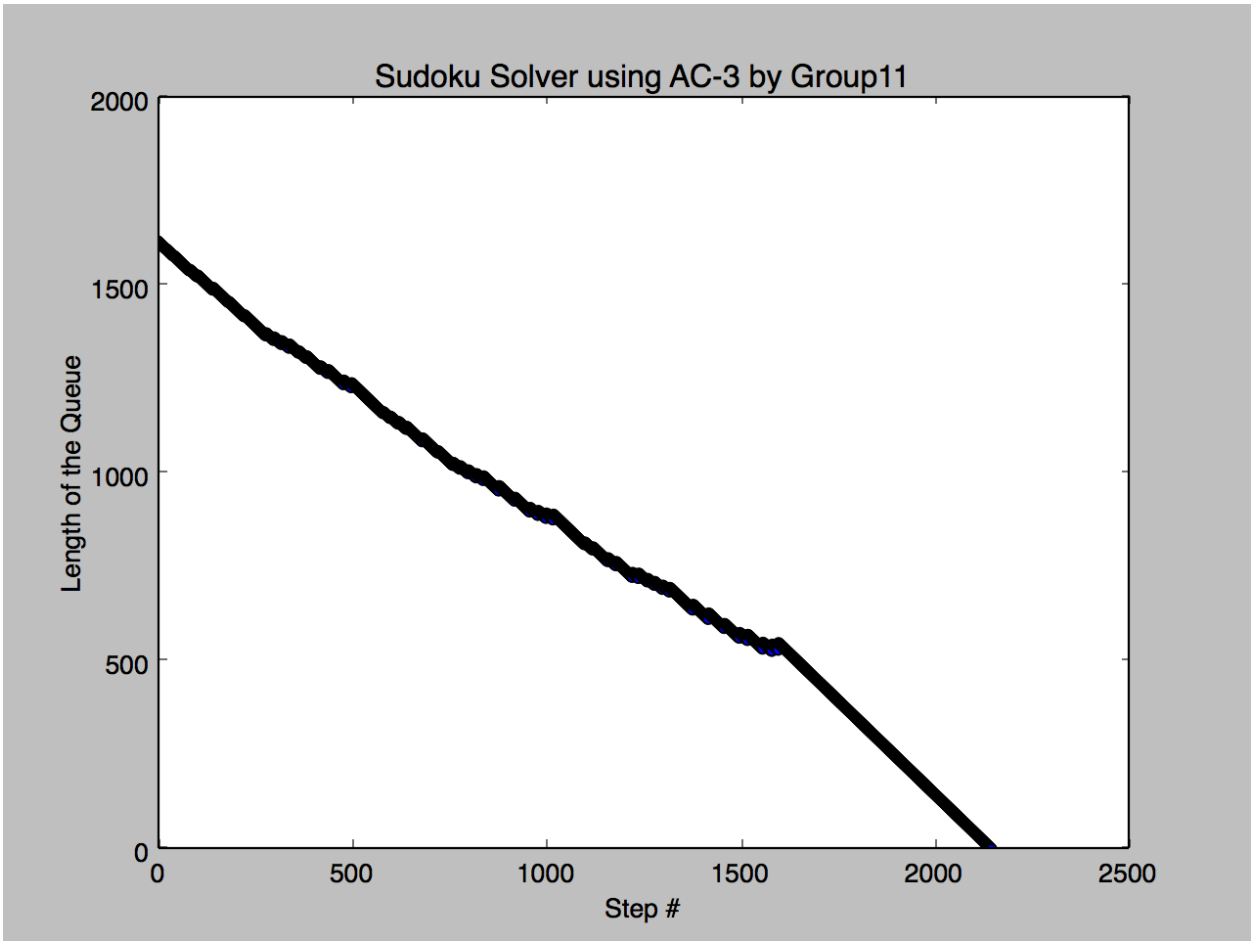
1 2 8 5 9 3 6 7 4

7 6 3 4 2 1 5 9 8

8 7 9 2 4 6 1 3 5

3 1 2 7 8 5 4 6 9

6 4 5 3 1 9 2 8 7



The number of steps taken by AC-3: 2150

Scenario 4

Given:

8 - - - - - - - -

- - 3 6 - - - - -

- 7 - - 9 - 2 - -

- 5 - - - 7 - - -

- - - - 4 5 7 - -

- - - 1 - - - 3 -

- - 1 - - - - 6 8

- - 8 5 - - - 1 -

- 9 - - - - 4 - -

Not Solved by AC-3

Solution:

8 1 2 7 5 3 6 4 9

9 4 3 6 8 2 1 7 5

6 7 5 4 9 1 2 8 3

1 5 4 2 3 7 8 9 6

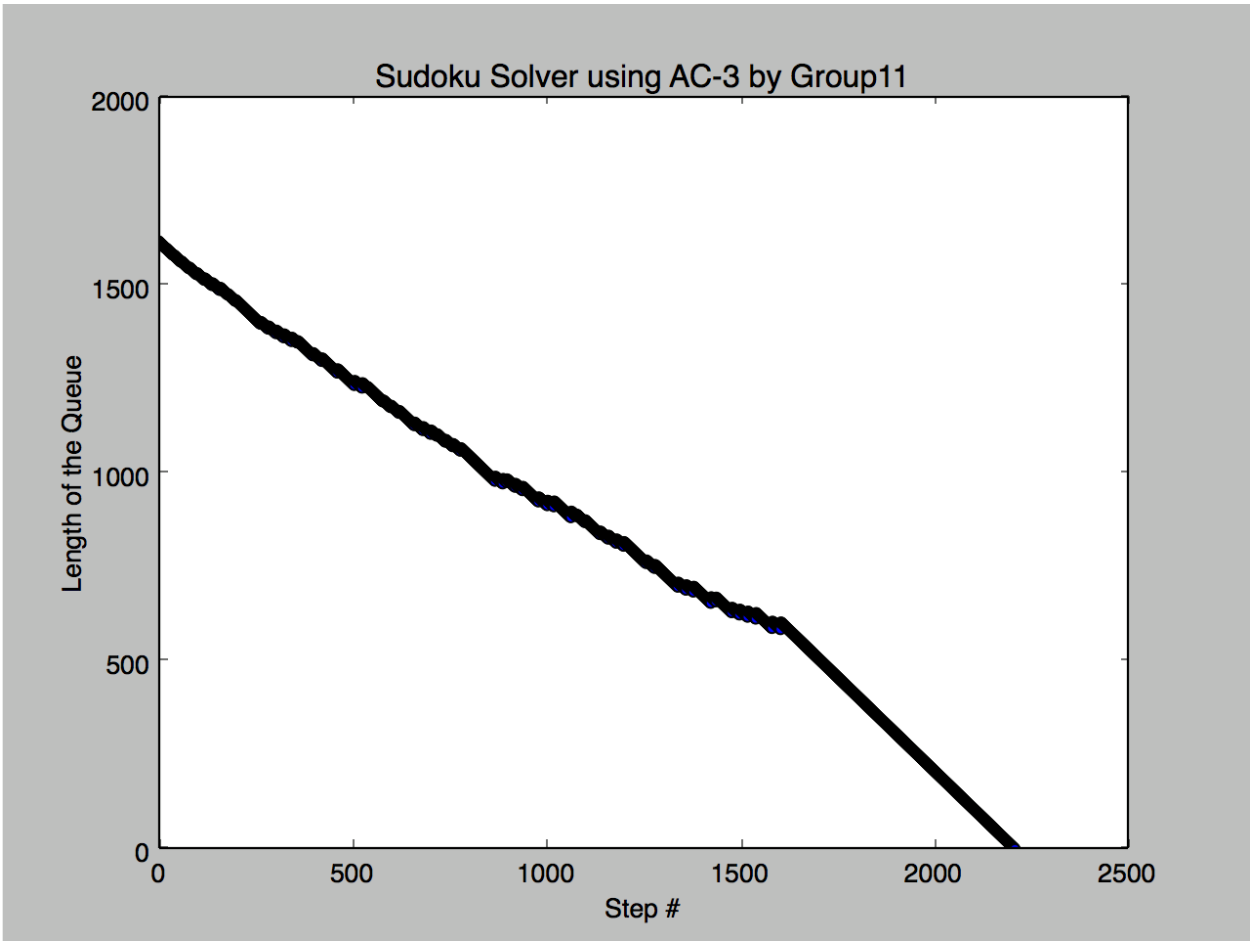
3 6 9 8 4 5 7 2 1

2 8 7 1 6 9 5 3 4

5 2 1 9 7 4 3 6 8

4 3 8 5 2 6 9 1 7

7 9 6 3 1 8 4 5 2



The number of steps taken by AC-3: 2212

Scenario 5

Given:

- - - - - - - - -

- - - - - 3 - 8 5

- - 1 - 2 - - - -

- - - 5 - 7 - - -

- - 4 - - - 1 - -

- 9 - - - - - - -

5 - - - - - - 7 3

- - 2 - 1 - - - -

- - - - 4 - - - 9

Not Solved by AC-3

Solution:

9 8 7 6 5 4 3 2 1

2 4 6 1 7 3 9 8 5

3 5 1 9 2 8 7 4 6

1 2 8 5 3 7 6 9 4

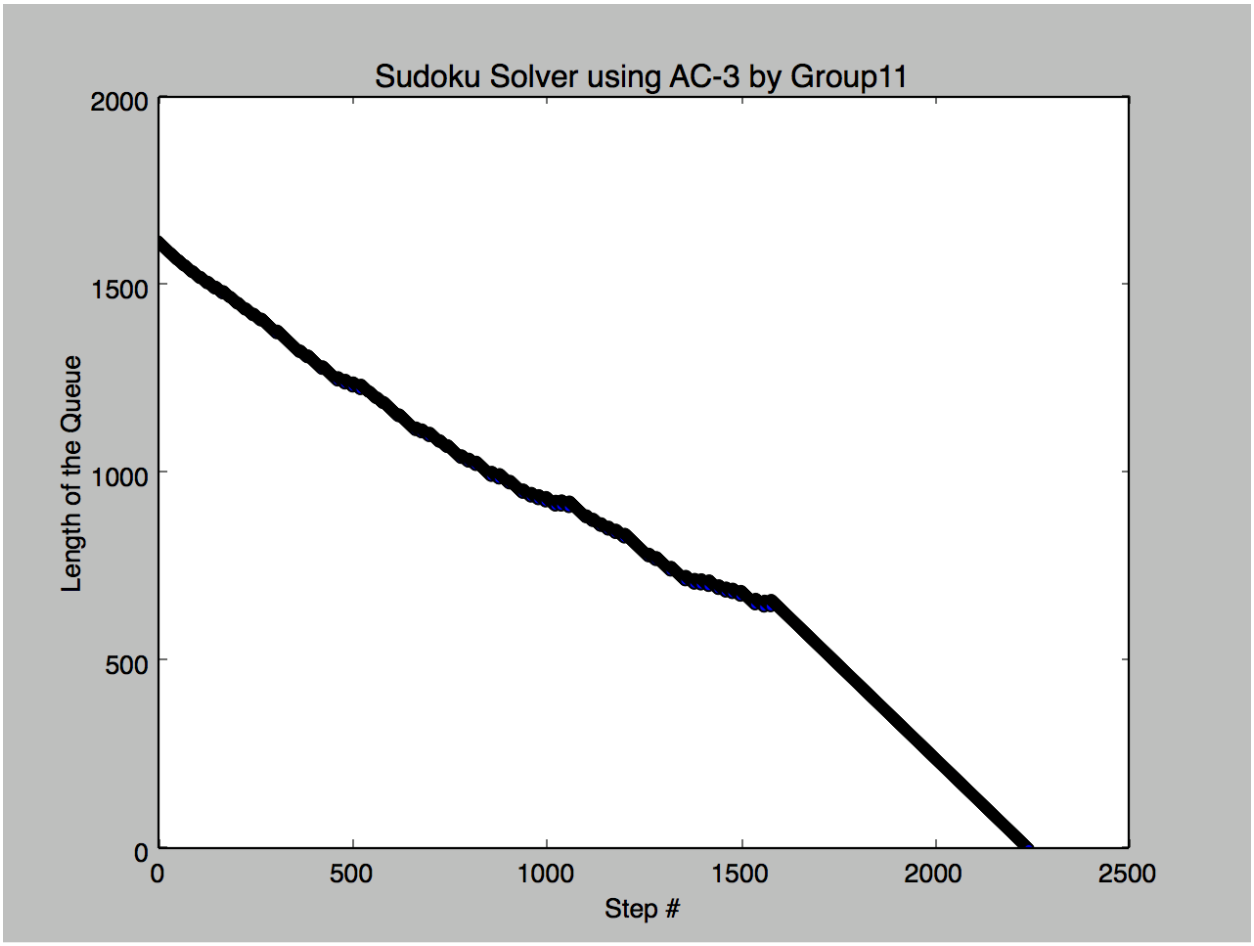
6 3 4 8 9 2 1 5 7

7 9 5 4 6 1 8 3 2

5 1 9 2 8 6 4 7 3

4 7 2 3 1 9 5 6 8

8 6 3 7 4 5 2 1 9



The number of steps taken by AC-3: 2246
