CSCE-496
Assignment 1
Brice Johnson
Due: 2016/1/29

Experiments

1)

I am represent my sudoku grid as a 2-d array. I have set up the initial grid with all 1's in row 1, all 2's in row 2, etc. I set up the grid in such a manner that the columns are fixed and have 1 - 9 already in each column.

My transform function is a simple swap. I choose one random column and two random rows and I swap the two row elements within that column. The neighborhood for each swap is the column, any two elements in a column can be swapped. An example would be to have column 4 and rows 2 and 7. You would swap the element of column 4, row 2 with he element in column 4, row 7. Then you check the fitness of the entire sudoku grid.

I am using two fitness functions, one for the row fitness and one for the sub square fitness. Both are set up identically in terms of the algorithm to calculate the fitness. Also I am minimizing the fitness. To calculate the fitness of each I start out with a fitness of 81. I look in each row and check if all 9 digit are present, if they are i subtract 9 from the total fitness. This means I take 1 off of the total fitness for each digit that appears once, repeated digits will only be counted once. At the end of the fitness function a 0 would mean that each digit is in each row exactly once. Any thing other than 0 would mean that a row is not correct. This is the same idea for the sub squares, except checking if each sub square has all 9 digits or not. If both the row and sub square fitnesses are 0, the sudoku problem is solved.

An example of calculating the fitness is seen below:

2, 9, 5 | 7, 2, 3 | 8, 6, 1
4, 3, 1 | 8, 6, 5 | 9, 2, 7
8, 7, 6 | 1, 9, 2 | 5, 4, 3

3, 8, 7 | 4, 5, 9 | 2, 1, 6
6, 1, 2 | 3, 8, 7 | 4, 9, 5
5, 4, 9 | 2, 1, 6 | 7, 3, 8

7, 6, 3 | 5, 4, 4 | 1, 8, 9
9, 2, 8 | 6, 7, 1 | 3, 5, 4
1, 5, 4 | 9, 3, 8 | 6, 7, 2

If you look at the two underlined digits, there are in the wrong place. When calculating the fitness for row 1, you would initially have the fitness 81, then subtract 1 for the first 2, the 9, 5, 7, 3, 8, 6, and 1 leaving you with a fitness of 73. You would continues down the rows until the seventh row when you would only subtract 8 from the total fitness. At the end of the row fitness calculation you would be left with a fitness of 2.

The same concept goes for the sub squares, in the second sub square, you would only subtract 8 from the total fitness, and again in the eighth subsquare, leaving you with a fitness of 2.

The fitnesses for this sudoku grid would be: row fitness - 2, sub square fitness - 2.

2)

I have two separate stopping criteria. The first is when both the row fitness and the sub square fitness are zero, this means the sudoku grid is solved. The second stopping criteria is a fixed number of iterations. This number is normally 1,000,000 for the first part of the assignment. Majority of the time a solution will be found within 1,000,000 iterations

3)

An efficient way of recalculating the fitness after each move for the rows (sub squares would be the same method, just with squares not rows) would require having the fitness for each row calculated at all times. Would would have an array (elements 1-9) that contained the fitness of that particular row. After the swap is made you would only need to recalculate the two row fitnesses that were effected. From there you can sum up the fitness array and determine if it is better or worse than the previous fitness calculated. This would mean you only have to go through three, nine elements arrays for each fitness calculation, not the nine arrays that my current fitness function implements.

4)

a)

Only better	Better or Equal to
Fail	5527
Fail	Fail
Fail	2397
Fail	Fail
Fail	3051
Fail	1581
Fail	5794
Fail	2584
Fail	1109
Fail	Fail

b) Better or Equal to

Probability	Iterations			
0.000	2774	Fail	3453	5712
0.0001	6144	2675	6187	2000
0.001	1915	8519	6500	4132
0.01	142670	23123	117054	1544

c) Only Better

Probability	Iterations			
0.000	Fail	Fail	Fail	Fail
0.0001	Fail	Fail	Fail	Fail
0.001	Fail	Fail	Fail	Fail
0.01	Fail	Fail	Fail	Fail

From these experiments I have gather interesting data. With the Hill Climbing algorithm that I have implemented for solving sudoku grids, the fitnesses must be able to move along a plateau. As seen in all of the “Only Better” experiments, they all failed because they could not move along a plateau. Also, the random chance to choose a bad move did effect the number of iterations, some of the time. As seen in 4.b the 1% probability had on average more iterations to solve the problem, but it also contained the lowest number of iterations (in 4.b) to solve the sudoku. From this, I can only determine that having a small chance (<= 0.001) of choosing a bad move is beneficial because with no probability you see that it has more times the solution fails to come up.

If I were going to give a set of parameters for this problem, I would have the new fitness to be Better or Equal To the previous best and have a small probability (<= 0.001) of choosing a bad move. These parameters seem to give a good balance between reliability (not failing) and quickness to a solution.