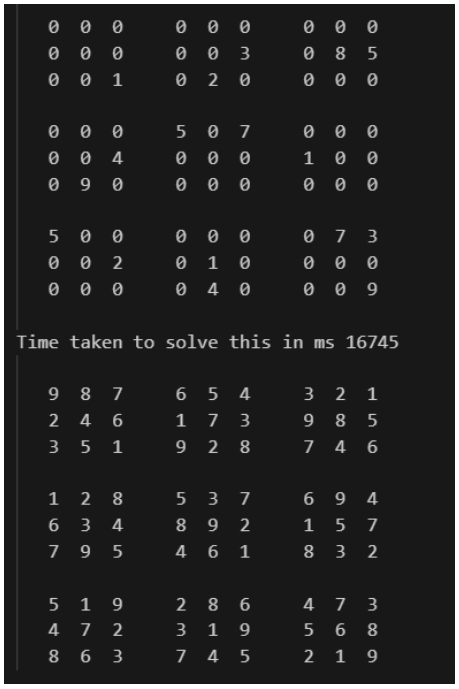
Determining the complexity of a sudoku solution based on the initial conditions

Abstract:

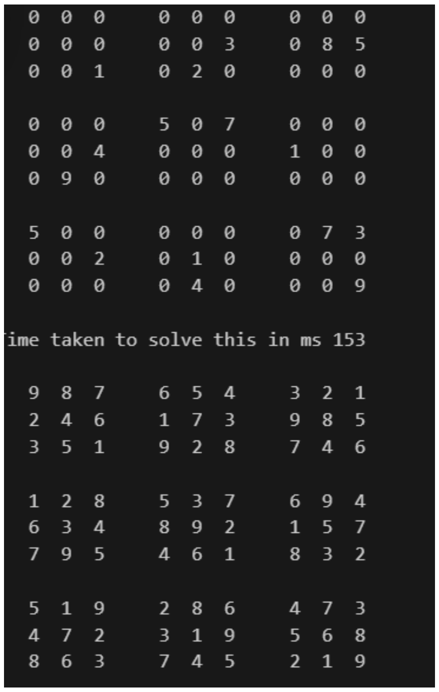
Sudoku is a well-known single-player game popularly played as a pastime. Its simplicity, yet the challenge of figuring out the values of the cells, makes it a really enjoyable across all levels of players. Based on the number of initial correct values provided, one can easily tune the difficulty of the game. Modern computers with their algorithms can very easily solve sudoku puzzles in an astonishingly short amount of time. This project focuses on a particular algorithm to solve a sudoku puzzle which is based on depth-based search in a stack data structure using the technique of backtracking. The project implemented various ideas to understand the systematic game and allowed for a variety of extensions. The key finding reported from the project is the almost parabolic dependence of time required to find the solution (if it exists) to the number of initial cells provided.

Results:

Firstly, the method of going serially across the board turned out to be extremely inefficient as expected. I defined a new method called findBetterCell() that figures out the next cell based on the minimum possible values it can have. That seemed natural and it worked better than the previous method. To test its efficiency, I created a board called “breakalgorithm.txt” in my codebase that can be run to see the difference.

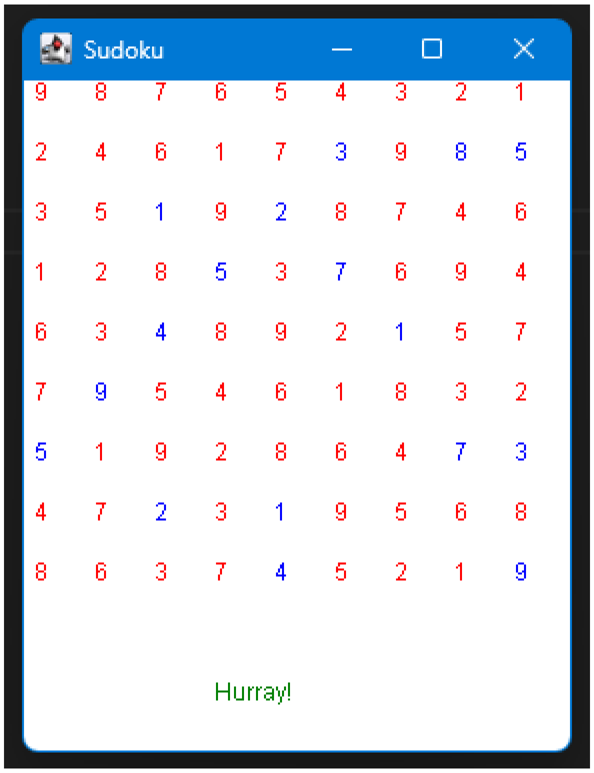


```
   0  0  0    0  0  0    0  0  0
   0  0  0    0  0  3    0  8  5
   0  0  1    0  2  0    0  0  0

   0  0  0    5  0  7    0  0  0
   0  0  4    0  0  0    1  0  0
   0  9  0    0  0  0    0  0  0

   5  0  0    0  0  0    0  7  3
   0  0  2    0  1  0    0  0  0
   0  0  0    0  4  0    0  0  9

Time taken to solve this in ms 16745

   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
```



```
  0  0  0    0  0  0    0  0  0
  0  0  0    0  0  3    0  8  5
  0  0  1    0  2  0    0  0  0

  0  0  0    5  0  7    0  0  0
  0  0  4    0  0  0    1  0  0
  0  9  0    0  0  0    0  0  0

  5  0  0    0  0  0    0  7  3
  0  0  2    0  1  0    0  0  0
  0  0  0    0  4  0    0  0  9

ime taken to solve this in ms 153

  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
```

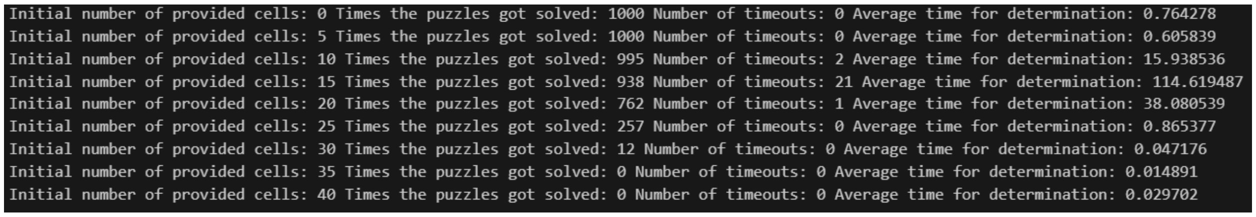
These images highlight the time taken to solve the same board using two algorithms. The findNextCell() is present to the left and the findBetterCell() is to the right. The difference in their time is astonishing.



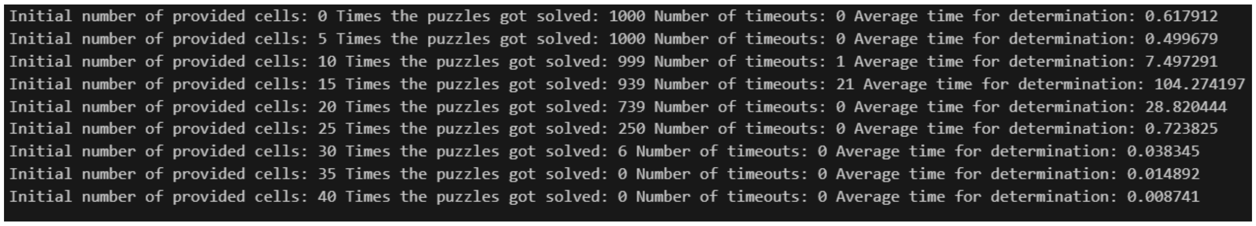
```
9  8  7  6  5  4  3  2  1
2  4  6  1  7  3  9  8  5
3  5  1  9  2  8  7  4  6
1  2  8  5  3  7  6  9  4
6  3  4  8  9  2  1  5  7
7  9  5  4  6  1  8  3  2
5  1  9  2  8  6  4  7  3
4  7  2  3  1  9  5  6  8
8  6  3  7  4  5  2  1  9

Hurray!
```

The GUI interface of the program looked like this.

Next, I tried to run a large number of games to understand the pattern of time consumption and number of initial values provided. The results of the programs are as following:



```
Initial number of provided cells: 0 Times the puzzles got solved: 1000 Number of timeouts: 0 Average time for determination: 0.764278
Initial number of provided cells: 5 Times the puzzles got solved: 1000 Number of timeouts: 0 Average time for determination: 0.605839
Initial number of provided cells: 10 Times the puzzles got solved: 995 Number of timeouts: 2 Average time for determination: 15.938536
Initial number of provided cells: 15 Times the puzzles got solved: 938 Number of timeouts: 21 Average time for determination: 114.619487
Initial number of provided cells: 20 Times the puzzles got solved: 762 Number of timeouts: 1 Average time for determination: 38.080539
Initial number of provided cells: 25 Times the puzzles got solved: 257 Number of timeouts: 0 Average time for determination: 0.865377
Initial number of provided cells: 30 Times the puzzles got solved: 12 Number of timeouts: 0 Average time for determination: 0.047176
Initial number of provided cells: 35 Times the puzzles got solved: 0 Number of timeouts: 0 Average time for determination: 0.014891
Initial number of provided cells: 40 Times the puzzles got solved: 0 Number of timeouts: 0 Average time for determination: 0.029702
```

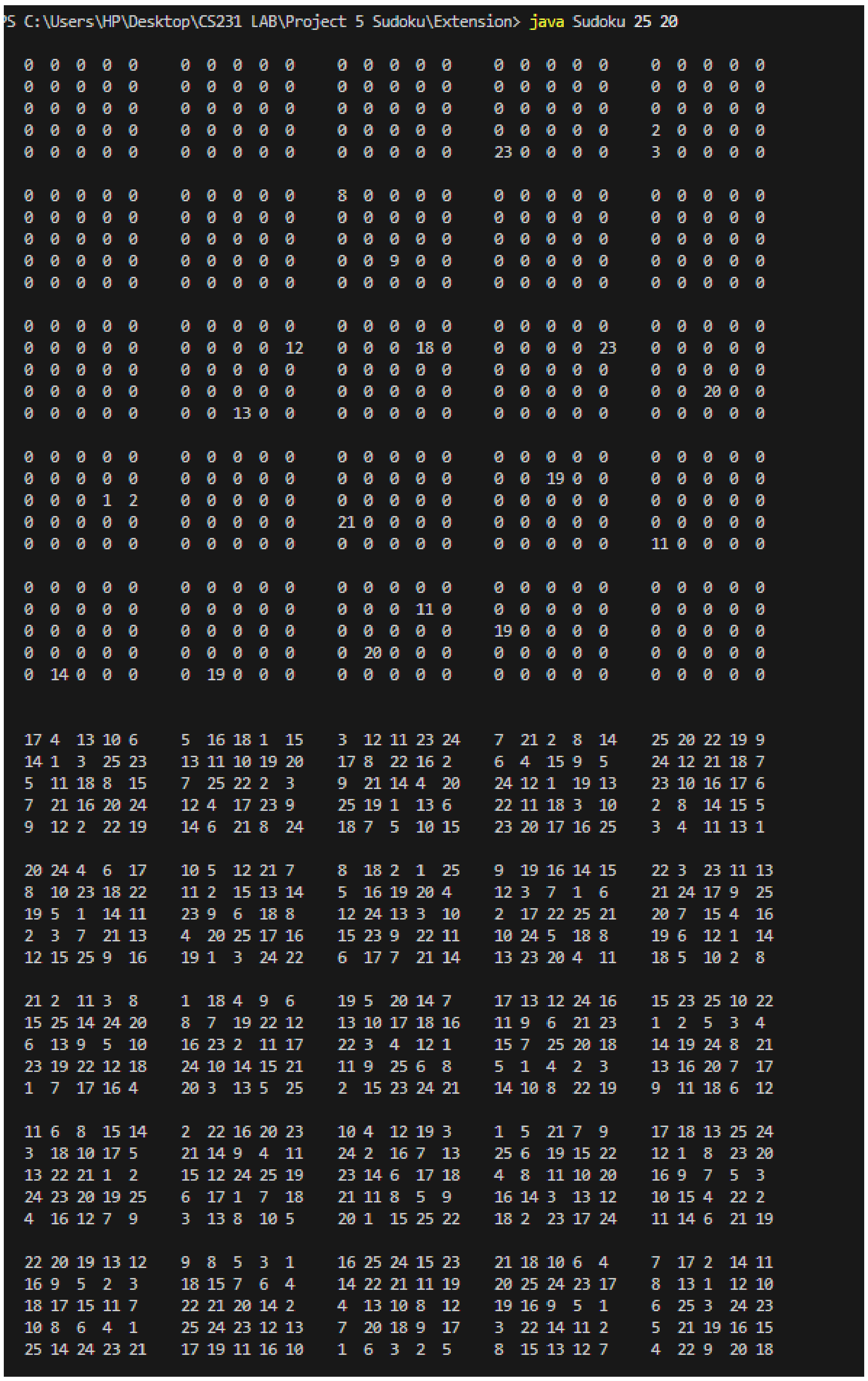


```
Initial number of provided cells: 0 Times the puzzles got solved: 1000 Number of timeouts: 0 Average time for determination: 0.617912
Initial number of provided cells: 5 Times the puzzles got solved: 1000 Number of timeouts: 0 Average time for determination: 0.499679
Initial number of provided cells: 10 Times the puzzles got solved: 999 Number of timeouts: 1 Average time for determination: 7.497291
Initial number of provided cells: 15 Times the puzzles got solved: 939 Number of timeouts: 21 Average time for determination: 104.274197
Initial number of provided cells: 20 Times the puzzles got solved: 739 Number of timeouts: 0 Average time for determination: 28.820444
Initial number of provided cells: 25 Times the puzzles got solved: 250 Number of timeouts: 0 Average time for determination: 0.723825
Initial number of provided cells: 30 Times the puzzles got solved: 6 Number of timeouts: 0 Average time for determination: 0.038345
Initial number of provided cells: 35 Times the puzzles got solved: 0 Number of timeouts: 0 Average time for determination: 0.014892
Initial number of provided cells: 40 Times the puzzles got solved: 0 Number of timeouts: 0 Average time for determination: 0.008741
```

Surprisingly, the time taken is maximum when the number of values provided is around 15-20. In hindsight, it makes sense because when the number of values is very less, there are a lot of possible solutions and when its too big, the number of solutions goes to 0 very quickly. So the program simply figures out its unsolvable leading to very low time even with valid initial boards.

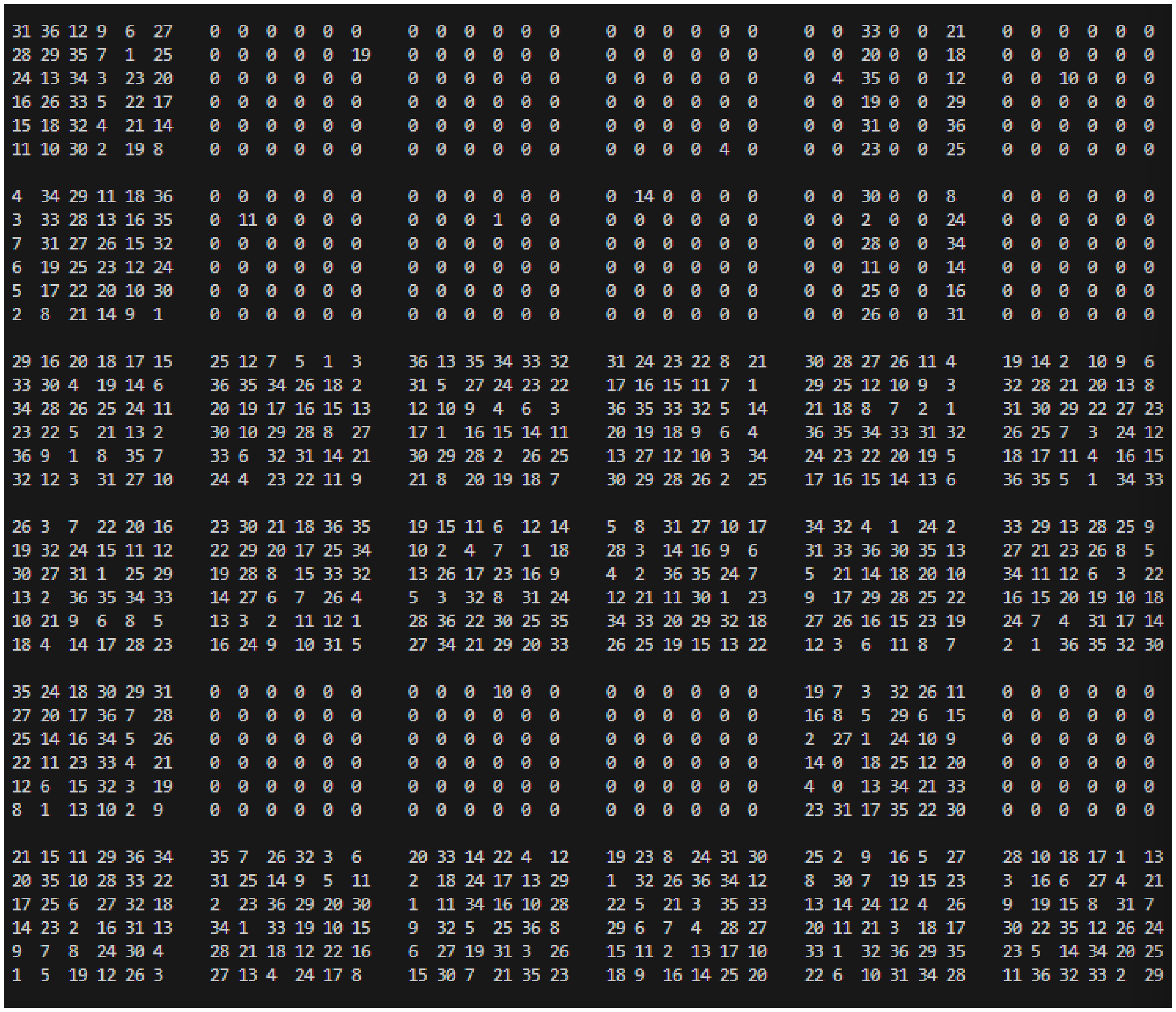
Extensions:

As an extension, I added the ability to add the size of the board and the number of initial values provided as arguments from the command line. The board size had to be a square to enable the logic of sudoku and the program using stacks is really inefficient for large board sizes.



```
S C:\Users\HP\Desktop\CS231 LAB\Project 5 Sudoku\Extension> java Sudoku 25 20

  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     2  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     23 0  0  0  0     3  0  0  0  0

  0  0  0  0  0     0  0  0  0  0     8  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  9  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0

  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  12    0  0  0  18 0     0  0  0  0  23    0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  20 0  0
  0  0  0  0  0     0  0  13 0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0

  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  19 0  0     0  0  0  0  0
  0  0  0  1  2     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     21 0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     11 0  0  0  0

  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  11 0     0  0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  0  0  0  0     19 0  0  0  0     0  0  0  0  0
  0  0  0  0  0     0  0  0  0  0     0  20 0  0  0     0  0  0  0  0     0  0  0  0  0
  0  14 0  0  0     0  19 0  0  0     0  0  0  0  0     0  0  0  0  0     0  0  0  0  0


  17 4  13 10 6     5  16 18 1  15    3  12 11 23 24    7  21 2  8  14    25 20 22 19 9
  14 1  3  25 23    13 11 10 19 20    17 8  22 16 2     6  4  15 9  5     24 12 21 18 7
  5  11 18 8  15    7  25 22 2  3     9  21 14 4  20    24 12 1  19 13    23 10 16 17 6
  7  21 16 20 24    12 4  17 23 9     25 19 1  13 6     22 11 18 3  10    2  8  14 15 5
  9  12 2  22 19    14 6  21 8  24    18 7  5  10 15    23 20 17 16 25    3  4  11 13 1

  20 24 4  6  17    10 5  12 21 7     8  18 2  1  25    9  19 16 14 15    22 3  23 11 13
  8  10 23 18 22    11 2  15 13 14    5  16 19 20 4     12 3  7  1  6     21 24 17 9  25
  19 5  1  14 11    23 9  6  18 8     12 24 13 3  10    2  17 22 25 21    20 7  15 4  16
  2  3  7  21 13    4  20 25 17 16    15 23 9  22 11    10 24 5  18 8     19 6  12 1  14
  12 15 25 9  16    19 1  3  24 22    6  17 7  21 14    13 23 20 4  11    18 5  10 2  8

  21 2  11 3  8     1  18 4  9  6     19 5  20 14 7     17 13 12 24 16    15 23 25 10 22
  15 25 14 24 20    8  7  19 22 12    13 10 17 18 16    11 9  6  21 23    1  2  5  3  4
  6  13 9  5  10    16 23 2  11 17    22 3  4  12 1     15 7  25 20 18    14 19 24 8  21
  23 19 22 12 18    24 10 14 15 21    11 9  25 6  8     5  1  4  2  3     13 16 20 7  17
  1  7  17 16 4     20 3  13 5  25    2  15 23 24 21    14 10 8  22 19    9  11 18 6  12

  11 6  8  15 14    2  22 16 20 23    10 4  12 19 3     1  5  21 7  9     17 18 13 25 24
  3  18 10 17 5     21 14 9  4  11    24 2  16 7  13    25 6  19 15 22    12 1  8  23 20
  13 22 21 1  2     15 12 24 25 19    23 14 6  17 18    4  8  11 10 20    16 9  7  5  3
  24 23 20 19 25    6  17 1  7  18    21 11 8  5  9     16 14 3  13 12    10 15 4  22 2
  4  16 12 7  9     3  13 8  10 5     20 1  15 25 22    18 2  23 17 24    11 14 6  21 19

  22 20 19 13 12    9  8  5  3  1     16 25 24 15 23    21 18 10 6  4     7  17 2  14 11
  16 9  5  2  3     18 15 7  6  4     14 22 21 11 19    20 25 24 23 17    8  13 1  12 10
  18 17 15 11 7     22 21 20 14 2     4  13 10 8  12    19 16 9  5  1     6  25 3  24 23
  10 8  6  4  1     25 24 23 12 13    7  20 18 9  17    3  22 14 11 2     5  21 19 16 15
  25 14 24 23 21    17 19 11 16 10    1  6  3  2  5     8  15 13 12 7     4  22 9  20 18
```

Here, using a 25*25 almost always works but 36*36 runs out of time even after 1 million iterations:



```
31 36 12 9  6  27    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  0  33 0  0  21    0  0  0  0  0  0
28 29 35 7  1  25    0  0  0  0  0  19    0  0  0  0  0  0     0  0  0  0  0  0     0  0  20 0  0  18    0  0  0  0  0  0
24 13 34 3  23 20    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  4  35 0  0  12    0  0  10 0  0  0
16 26 33 5  22 17    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  0  19 0  0  29    0  0  0  0  0  0
15 18 32 4  21 14    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  0  31 0  0  36    0  0  0  0  0  0
11 10 30 2  19 8     0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  4  0     0  0  23 0  0  25    0  0  0  0  0  0

4  34 29 11 18 36    0  0  0  0  0  0     0  0  0  0  0  0     0  14 0  0  0  0     0  0  30 0  0  8     0  0  0  0  0  0
3  33 28 13 16 35    0  11 0  0  0  0     0  0  0  1  0  0     0  0  0  0  0  0     0  0  2  0  0  24    0  0  0  0  0  0
7  31 27 26 15 32    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  0  28 0  0  34    0  0  0  0  0  0
6  19 25 23 12 24    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  0  11 0  0  14    0  0  0  0  0  0
5  17 22 20 10 30    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  0  25 0  0  16    0  0  0  0  0  0
2  8  21 14 9  1     0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     0  0  26 0  0  31    0  0  0  0  0  0

29 16 20 18 17 15    25 12 7  5  1  3     36 13 35 34 33 32    31 24 23 22 8  21    30 28 27 26 11 4     19 14 2  10 9  6
33 30 4  19 14 6     36 35 34 26 18 2     31 5  27 24 23 22    17 16 15 11 7  1     29 25 12 10 9  3     32 28 21 20 13 8
34 28 26 25 24 11    20 19 17 16 15 13    12 10 9  4  6  3     36 35 33 32 5  14    21 18 8  7  2  1     31 30 29 22 27 23
23 22 5  21 13 2     30 10 29 28 8  27    17 1  16 15 14 11    20 19 18 9  6  4     36 35 34 33 31 32    26 25 7  3  24 12
36 9  1  8  35 7     33 6  32 31 14 21    30 29 28 2  26 25    13 27 12 10 3  34    24 23 22 20 19 5     18 17 11 4  16 15
32 12 3  31 27 10    24 4  23 22 11 9     21 8  20 19 18 7     30 29 28 26 2  25    17 16 15 14 13 6     36 35 5  1  34 33

26 3  7  22 20 16    23 30 21 18 36 35    19 15 11 6  12 14    5  8  31 27 10 17    34 32 4  1  24 2     33 29 13 28 25 9
19 32 24 15 11 12    22 29 20 17 25 34    10 2  4  7  1  18    28 3  14 16 9  6     31 33 36 30 35 13    27 21 23 26 8  5
30 27 31 1  25 29    19 28 8  15 33 32    13 26 17 23 16 9     4  2  36 35 24 7     5  21 14 18 20 10    34 11 12 6  3  22
13 2  36 35 34 33    14 27 6  7  26 4     5  3  32 8  31 24    12 21 11 30 1  23    9  17 29 28 25 22    16 15 20 19 10 18
10 21 9  6  8  5     13 3  2  11 12 1     28 36 22 30 25 35    34 33 20 29 32 18    27 26 16 15 23 19    24 7  4  31 17 14
18 4  14 17 28 23    16 24 9  10 31 5     27 34 21 29 20 33    26 25 19 15 13 22    12 3  6  11 8  7     2  1  36 35 32 30

35 24 18 30 29 31    0  0  0  0  0  0     0  0  0  10 0  0     0  0  0  0  0  0     19 7  3  32 26 11    0  0  0  0  0  0
27 20 17 36 7  28    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     16 8  5  29 6  15    0  0  0  0  0  0
25 14 16 34 5  26    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     2  27 1  24 10 9     0  0  0  0  0  0
22 11 23 33 4  21    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     14 0  18 25 12 20    0  0  0  0  0  0
12 6  15 32 3  19    0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     4  0  13 34 21 33    0  0  0  0  0  0
8  1  13 10 2  9     0  0  0  0  0  0     0  0  0  0  0  0     0  0  0  0  0  0     23 31 17 35 22 30    0  0  0  0  0  0

21 15 11 29 36 34    35 7  26 32 3  6     20 33 14 22 4  12    19 23 8  24 31 30    25 2  9  16 5  27    28 10 18 17 1  13
20 35 10 28 33 22    31 25 14 9  5  11    2  18 24 17 13 29    1  32 26 36 34 12    8  30 7  19 15 23    3  16 6  27 4  21
17 25 6  27 32 18    2  23 36 29 20 30    1  11 34 16 10 28    22 5  21 3  35 33    13 14 24 12 4  26    9  19 15 8  31 7
14 23 2  16 31 13    34 1  33 19 10 15    9  32 5  25 36 8     29 6  7  4  28 27    20 11 21 3  18 17    30 22 35 12 26 24
9  7  8  24 30 4     28 21 18 12 22 16    6  27 19 31 3  26    15 11 2  13 17 10    33 1  32 36 29 35    23 5  14 34 20 25
1  5  19 12 26 3     27 13 4  24 17 8     15 30 7  21 35 23    18 9  16 14 25 20    22 6  10 31 34 28    11 36 32 33 2  29
```

I also implemented a ShowViz element to my Sudoku constructor depending on the need to see the graphical demonstration, but found printing to terminal better.

In general, using larger boards leads to inefficient solution but due to limitation of computing power I could not verify with actual examples. I also included ability to read from txt files, tried using “sixteen.txt” in my case to read any board size.

Acknowledgements:

In completing this project, I consulted with Max and Allen, but did not look at any code. I also consulted the wikipedia page on the Sudoku game and board where the algorithm does not work