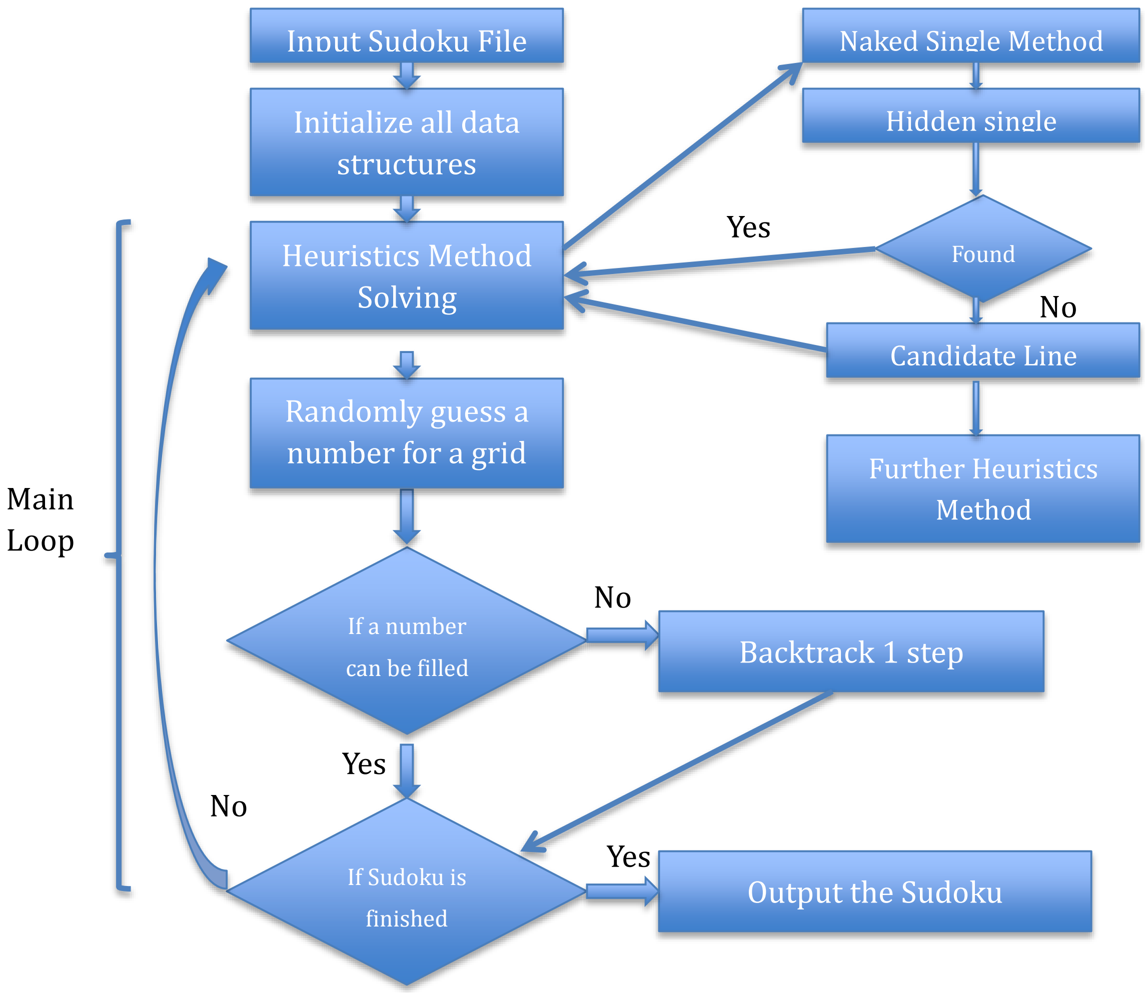
A Heuristics-Based, Depth-First-Search Sudoku Solver

By Wang Duo
CID No. 00737098
22nd March 2013
Electronic and Information Engineering 1st year

1. Introduction

Most of the Sudoku solver today use depth-first search(DFS) algorithm to search for the solution. The algorithm, though simple to write, is relatively slow. Therefore, this Sudoku solver aims to combine several heuristics methods of solving Sudoku together with the depth-first search(DFS) algorithm to increase the speed of solver. Moreover, the solver can actually be used to evaluate the difficulty of the Sudoku for human solver because this solver uses an expert system that emulates human player's method to solve Sudoku.

2. Overview of the Solver Structure



The solver uses a loop that terminates when the Sudoku is finished. In the loop the solver first try the three heuristics methods. It is possible to put many more heuristics method in the software, however due to time constraints, only three most basic methods are used. The three methods will be explained in later section. If all three fails, a random number is guessed for the first grid with the least number of possibilities. The solver will then use heuristics methods again to search. If a solution cannot be found and it is not possible to make any guesses, the solver backtrack one step to try other possibilities for previous guess. If the solver could not find a solution, it will simply output " The sudoku is not solvable" on the standard output.

3. Heuristics Methods

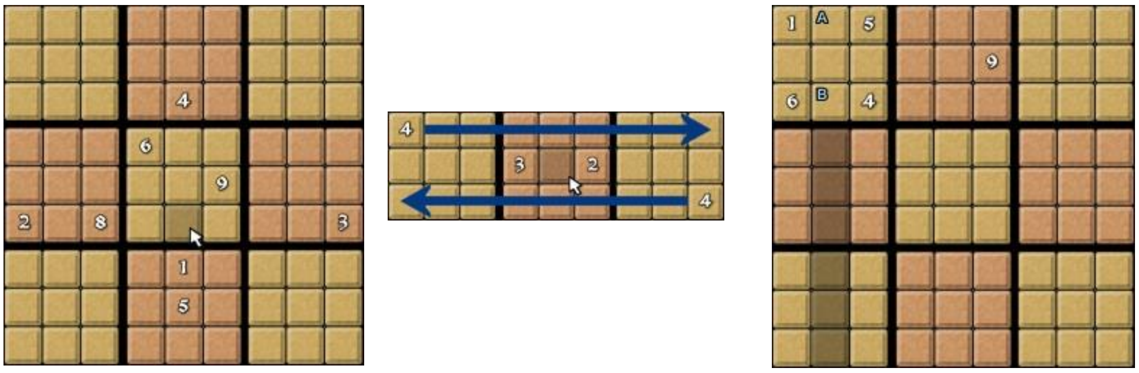


Fig. 1 a.) Naked Single Method b.) Hidden Single Method c.) Candidate Line Method

3.1 Naked single method

In figure 1 a.), the grid clicked by the cursor can only be filled by number "7" because all other number has already appeared in the row, column or block the grid is in.

3.2 Hidden single method

In figure 1 b.), only number "4" can be filled into the grid clicked by the cursor because the two "4"s in the other two blocks eliminate the two 4's eliminate two rows and two blocks, leaving a possible three cells in the middle for our final 4, two of the middle cells have been taken up by a 3 and 2 two, so its logical that the final place must be our final 4.

3.3 Candidate Line Method

The number 9 here has forced block 1 to either house a 9 in cell A or B, this in turn allows the shadowed column to remove all candidate 9's from each cell.

4. Major Data Structures

Matrix[i][j]:
A data structure that records the filled Sudoku matrix.

Spec[i][j]:

A structure recording the possible numbers can be filled in (i,j) position. Spec[i][j].count records the number of possible values in the grid while spec[i][j].list[k] records whether the number “k” can be filled in this grid.

rec.row[i][j]/column[i][j]/block[i][j]:
A structure recording number of grids that number “j” can be filled in for row, column or block “I”.

5. Random Number Guessing and Backtracking

If all three heuristics methods fail, the algorithm will randomly try a number in the first found grid with least number of possibilities. Then the number tried for that grid is recorded in an array. If the solver tracks back to this step of guessing from a dead ended search, the next guessed number must be greater than the recorded number if the same grid is guessed. After guessing, the solver will use heuristics methods again to solve the Sudoku until another number needs to be guessed or the solution is found. If solution is not found and no other number can be guessed for any grid the solver will record zero in the array of guessed number and backtrack to the previous step of guessing to try the next smallest number for that step.

6. Input and Output

The solver input Sudoku from a file named”sudoku_x.txt” where “x” is a number. The user will be prompted to input an “x” using standard input. The solver will output the finished Sudoku into a file named “solution.txt”.

7. Performance Analysis

Six test cases are provided together with the source code. Test cases 1 and 3 are simple sudoku which takes 0.297 ms and 0.323ms. Test case 2 is difficult which takes 0.344ms to solve. Test cases 4 and 5 are ranked as master level Sudoku. The solver takes 2.748ms and 1.166ms to solve respectively. Test case 6 is called “One of the most difficult Sudoku ever”, which takes the solver 209.74ms to solve it. All tests are done with an Intel i5 CPU. Generally this solver performs well as it is much faster than most of the backtracking algorithm. It is even faster than Khuth’s Algorithm X that takes 21.9ms to solve test case 5. However the speed of the solver can be further increased if more logical methods are coded into the program.

8. Conclusion

This solver combines the heuristics methods with backtracking algorithm to produce a very fast Sudoku solver. The solver is also useful for evaluating the difficulty level of the Sudoku for human solvers.