Progress Report

Bao Yelun, 2014012710
Liang Zhenxiao, 2014012508

January 14, 2018

1 Introduction

Sudoku, originally called Number Place, is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution. Completed games are always a type of Latin square with an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column, or any of the nine 3×3 subregions of the 9×9 playing board.

In this project we want to design an AI engine to play Sudoku game. The Sudoku will not only be the classic 9×9 playing board, but also bigger size Sudoku and Sudoku with irregular cells, by introducing new algorithms.

2 Algorithm Description

The input of a Sudoku game is a digit sequence with totally 81 digits, each of which represents the given number in the puzzle, from left to right, top to bottom. We will use 0 to represent the empty grids to be fill out. And the expected solution, the output, is also a 81-digit sequence, which is the right digit assignment satisfying the rules. There is no 0 in the output.

We are going to develop some new algorithms to speed up the solving process. Based on back tracing algorithm, we assign the nonempty position's value to the initial dictionary variable, which is the table of our playing board. When we meet 0, assign the list $[1, 2, 3, 4, 5, 6, 7, 8, 9]$ to the entry. Then eliminate the digits in each grid which violate the constraint rules in current state. If a grid's list remains only one digit, assign it the grid and update the current state. Naked Twin or Triplets trick, which is also called Naked Twin exclusion rule, is used, in order to speed up this process. Even if the pair of grids are not assigned, we can also use these naked twins (or triplets) to eliminate some candidates.

Implement recursion techniques to create depth-first-search trees for solving hard sudokus that need guess work in any particular box to proceed. If we can find a feasible solution, just halt and return this solution.

3 Advanced Variation

In the search process we need guess work in any particular box to proceed. Here the right choice, or reasonable attempt priority is very important to our solver, which can help us find a feasible solution much faster. Therefore, a reasonable and novel heuristic function, instead of simple natural number order, is a very good tool to decide the order of trial. In this step we do not find a very efficient heuristic function, but we will design one in the following days.

Moreover, we will also try to solve Sudoku with bigger size. The advantage of implementing heuristic function will be more conspicuous with the growth of the size. We will also try to solve Sudoku with irregular cells. The performance of the new algorithm can be measured by the time it consumes to solve the testing set problems, compared to that consumed by the simple back tracing solver. This evaluation also involves the design of larger Sudoku, since this data set is hard to find on the Internet.

4 Current Results

Now we have completed the basic back tracing algorithm (which can be seen in util.py). We can run the python file and type the input (only 81 digits), and then we can get the puzzle solved. This algorithm will be add with much other novel tricks, and will be generalized to a more universal solver.

```
❯ python3 main.py
type sudoku as a long string of 81 characters with 0 as unsolved place
004300209005009001070060043006002087190007400050083000600000105003508690042910300
8 6 4 |3 7 1 |2 5 9
3 2 5 |8 4 9 |7 6 1
9 7 1 |2 6 5 |8 4 3
------+------+------
4 3 6 |1 9 2 |5 8 7
1 9 8 |6 5 7 |4 3 2
2 5 7 |4 8 3 |9 1 6
------+------+------
6 8 9 |7 3 4 |1 2 5
7 1 3 |5 2 8 |6 9 4
5 4 2 |9 1 6 |3 7 8
```