Sudoku - Low Level Programming

Khanh Duy NGUYEN TU M1 Cyber

January 2023

Contents

1 Introduction

In this Sudoku project, we create a software that solve and generate sudoku puzzles. Apart from traditional 9x9 sudoku, this software works on multiple size from 1x1 to 64x64.

1.1 Terminology

- Grid : The sudoku's board
- Sub-grid : A set of cells in a row, column or block
- Candidate : Possible value in a cell
- Given : Filled cell

2 Solver

For the solver, we combine Pencil and Paper method which list all candidates for each cell then try to reduce it, and brute force which test all candidates until the right one.

2.1 Heuristics

Heuristics are techniques that help narrow down candidates for each cell. All techniques introduced below act on a sub-grid.

2.1.1 Cross Hatching

Cross hatching is the most basic and useful technique. In a sub-grid, if a cell is given, we can then eliminate that cell's value from all other cells.

```
function cross_hatching(subgrid)
    to_remove = empty set of value
    for each cell in subgrid
        if cell is given
            add cell's value to to_remove
    for each cell in subgrid
        if cell is not given
            remove candidates that appear in to_remove
```

2.1.2 Lone Number

In a sub-grid, if there is a value that only appear in one cell, that value must be set in that cell.

```
function lone_number(subgrid)
    appeared = empty set of value
    repeated = empty set of value
    for each cell in subgrid
        add cell's candidates to appeared
        if value already exists in appeared
            add value to repeated
    lone = set of value that in appeared but not repeated
    for each cell in subgrid
        if cell is not given
            if cell has candidate in lone
                remove other candidates
```

2.1.3 Naked Subset

In a sub-grid, if there are N cells that have the same(or partly) N candidates, these candidates can be eliminates from other cells.

```
function naked_subset(subgrid)
    for each cell1 in subgrid
        count = 0
        for each cell2 in subgrid
            if cell2's candidates is part or equal cell1's
                count++
        if count == number of cell1's candidates
            remove cell1's candidates from other cells
```

2.1.4 Hidden Subset

In a sub-grid, if there is a subset of N candidates that only appear in N cells. Then these candidates can only be in these cells, thus eliminates all other candidates in these cells.
This seems hard at first glance but very simple when we look at another angle. Hidden Subset is very similar to Naked Subset to recognize. Instead of finding "a subset of N candidates only appear in N cells", we find "a subset of N cells only contain N candidates".
Knowing this, at the beginning of Hidden Subset, I change sub-grid from a set of N cells, each contain list of candidates to a set of N candidates, each contain list of cells it appear in. For example:
Cell

- 1 : 123
- 2 : 34
- 3 : 14
- 4 : 123

Will become:
Candidate

- 1 : 134
- 2 : 14
- 3 : 124
- 4 : 23

And from this set, we apply Naked Subset logic to find the set of candidates.

2.1.5 Heuristics Algorithm

We first keep applying Cross Hatching and Lone Number to every sub-grid until it cannot be changed anymore. We then apply Naked Subset and Hidden Subset. If it's effective, we go back and apply Cross Hatching and Lone Number and continue this loop until all heuristics fail.

```
function subgrid_heuristics(subgrid, level)
    if level
        apply naked_subset, hidden_subset
    else
        apply cross hatching, lone number

function grid_heuristics(grid)
    level = 0
    while level < 2
        for each subgrid
            subgrid_heuristics(subgrid, level)
        if nothing changes
            level++
        else
            level-- /* cap at 0 */
```

2.2 Solver Algorithm

We begin by checking consistency of the grid, then applying heuristics, then make a choice, create new grid by applying the choice then call solver on this new grid. If we can solve this grid, return. If not, discard the choice from the original grid then make another choice.

```
function solver(grid)
    if grid is not consistent
        return NULL
    if grid is solved
        return grid
```

```
make a choice
while choice is not empty
    new_grid = copy(grid)
    apply choice to new_grid
    result = solver(new_grid)
    if result
        return result
    discard choice
    make new choice
```

3 Generator

3.1 Algorithm

Start with an empty grid, we fill the first row randomly. Then we apply the solver on this grid until it is solved. The next step is base on mode: For normal mode, we randomly remove 40% of the values For unique mode, we do the same but in the same time applying solver after each remove to ensure that there is an unique solution for the grid, if not we remove other cell.

```
function generator(size, mode)
    create_grid(size);
    fill randomly first row(grid);
    grid = grid_solver(grid, mode_first);
    if mode_first
        remove 40% cells
    else
        for each step of remove 40% cells
            grid_solver(grid)
            if number of solutions != 1
                unremove cell
```

4 Optimization

4.0.1 Make choice

Before, to make a choice, we find the cell with the least candidates. This will get expensive as the grid size get bigger since we have to search through all the cells. That's why I have decided to choose the first not empty cell.
Which candidate to choose is also need to consider, in mode "–all" this doesn't seem to be a problem because we will check all the candidate anyway, but in mode first and generator, we can improve performance if we can choose the right candidate faster. That's why instead of choosing the leftmost candidate I choose randomly. In the future, we can improve this more by choosing the candidate that had appeared the least in the grid.

4.0.2 Order of heuristics

I apply heuristics in order that base on their complexities:

- Cross Hatching : $O(N)$
- Lone Number : $O(N)$
- Naked Subset : $O(N^2)$
- Hidden Subset : $O(N^2)$

Since Naked Subset and Hidden Subset are both expensive, We don't want to call them alot, We only call them when we are sure that Cross Hatching and Lone Number can't do anything. That's why we implement "level" into subgrid_heuristics

4.0.3 Good practices

- Use prefix operator (++i) instead of postfix operator (i++)
- If can, use bitwise operator