Sudoku Puzzle Solver

Xiao Lyu xxl160230

Abstract

The Sudoku puzzles are proven to be NP- complete [1]. And for 9 * 9 Sudoku puzzle problem, there are about 6.671 * 10^{21} valid Sudoku grids [2] . So enumerating every possible might be time consuming. Nowadays, we can use many computer algorithms to solve it efficiently. In this paper, we regard 9 * 9 Sudoku puzzle problem as constraint satisfaction problem. And then applying CSP search algorithms (constraint propagation) to solve it [3]. For this constraint satisfaction problem, I am using backtracking search and minimum remaining values heuristic.

1. Introduction

Sudoku puzzle is a puzzle with 9 * 9 grid. Each squares in the grid must contain one digit from 1 to 9. Every time, the puzzle is given at least 17 clues [4] . The goal is to fill other ungiven squares with digit 1 to 9 such that every row, every column and nine 3 * 3 boxes) contains one digit chosen from 1 to 9. An example is shown below.

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

(a) This is a Sudoku puzzle

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

(b) This is a solution of this Sudoku puzzle

Here are some definitions of Sudoku:

Unit : A row, column, or box is called a unit [3] . Every square has exactly 3 units [5] . A unit should be filled with digit 1 to 9. And each digit must show once in every unit.

Peer: Other squares that share same row or same column or same box with a certain square. Every square has 20 peers. Each square must have a different value from any of its peers [5] .

2. Related work

These years, various methods are applied to solve Sudoku puzzle problem, such as Graph theory [6], Artificial intelligence [7], and Genetic algorithm [8].

With respect to computer algorithm, backtracking, stochastic search, constraint programming are techniques to solve Sudoku puzzles.

Backtracking : Backtracking is a type of brute force search [9]. Brute force calculation constructing all possible answers, and seeing which ones really do work [2]. Backtracking is a depth-first search. Expanding an ungiven square and choosing one digit for this square one time. When the left squares meet contradiction, backtracking and choose another unchosen digit. Repeating this process until we get a solution for the puzzle.

This algorithm is simple but relatively slow compared to other algorithms because of its amount of computing.

Stochastic search : Using Stochastic search algorithm to try and discover what types of instances is best suited for [10]. This method includes simulated annealing and genetic algorithm.

Constraint programming : Sudoku can also regard as a constraint satisfied problem [11]. Combining with backtracking search could give a fast solution for all the Sudoku puzzles. Dancing links is a technique suggested by Donald E. Knuth to find all solutions to the exact cover problem [12].

3. Approach

I am using constraint propagation and backtracking search. Using minimum-remaining-values(MRV) heuristic to choose unassigned squares. Because this method is very fast and can give solution to all solvable Sudoku puzzles.

Regarding Sudoku puzzle problem as a CSP with 81 variables. Every square is a variable. We use the variable names A1 through A9 for the top row from left to right, down to I1 to I9 for the bottom row [3]. The domain of the empty squares is {1, 2, 3, 4, 5, 6, 7, 8, 9}, and the domain of given value squares is the value that has been given.

And the constraints are 27 Alldiff constraints, nine for each row, nine for each column and nine for each box. These constraints can be represented by two strategies:

(1) If a square has only one possible values, then eliminate that value from the square's peers.

(2) If a unit has only one possible place for a value, then put the value there. [5]

Using backtracking search combined with constraint propagation to solve Sudoku puzzle. Every time, using minimum-remaining-values(MRV) heuristic to choose an unassigned square and then assign a valid value for the square. Then applying constraint propagation to minimize the number of valid values for every square. This could reduce the amount of computation. If any contradiction happens, backtracking and assign the chosen square with other valid value.

The specific method of my Sudoku Puzzle Solver is below:

(1) Input a text file with "open file" button or input the puzzle by typing in the textfield. Every time could input one Sudoku puzzle with certain format. "0" or "." is stands for the empty square. And digit 1-9 stands for the given clue.
(2) Creating an empty grid with every square has the domain {1, 2, 3, 4, 5, 6, 7, 8, 9}.
(3) Getting the input puzzle and using constraint propagation to narrow down the domain of those given value squares and their peers.
(4) Using backtracking search to solve undecided squares. Applying constraint propagation in every trial.
(5) If there exists a solution, the solver could give a solution.

4. Implementation

This project consists of three files called Grid.java, SudokuGUI.java and Solver.java.

(1) Grid.java : This part is a data structure to store valid values for all squares and the current state of the puzzle.

    Storing valid values in a HashMap<String, String>, the key is the variable name such as "A1" and "B2". The value is the domain of corresponding key. At first, all the domain are initialized as {1, 2, 3, 4, 5, 6, 7, 8, 9}.

(2) SudokuGUI.java : This part is a graphical user interface for my Sudoku Puzzle Solver.

    Running this file, we could have a GUI for Sudoku solver.

    There is a 9 * 9 grid to display a puzzle that user inputs or the solution of the input puzzle. The "open file" button in the right and the textfield below allows user to input the puzzle. When using "open file" button, only text file is valid. And the input must have 81 valid character, 1-9 stands

for the given values, “0” or “.” stands for the empty square. Other characters are allowed, but the number of valid character must be 81.

Clicking “Solve” button, we could get the solution of the current puzzle.

(3) Solver.java : This part contains definition of Sudoku puzzle and the functions that implements backtracking searching and constraint propagation. Some functions are listed below:

a) checkValidGrid : check the validation of input.

b) gridValues : get the values from the input.

c) parseGrid : assign the puzzle to the grid.

d) displayGrid : display a grid.

e) assign : assign a value and then narrow down other peers domain

f) eliminate : eliminate a certain value from a square

g) search : backtracking searching until puzzle solved or meet contradiction

h) copy : copy a grid. It’s for backtracking when contradiction occurs.

i) mrv : choose the square using minimum remaining value heuristic. It is also for the process of backtracking searching.

j) giveSolution : give a solution of a puzzle.

k) timeRecord : give the time of solving process. It is for the testing.

5. Experiment and result

During the experiment, if we use only constraint propagation, we could only solve easy Sudoku puzzles but not those difficult ones. But combining with backtracking search, we could give the solution of all the solvable Sudoku puzzle.

Using the test data given by Peter Norvig [5] and the timeRecord function, we can get the time for solving Sudoku with different difficulties below:

```
This is the result of 50 easy problems:
Solved 50 problems.
Solved in 0.339 seconds.
Solved every problem in average 0.00678 seconds.
--------------------------------------------
This is the result of 11 hardest problems:
Solved 11 problems.
Solved in 0.037 seconds.
Solved every problem in average 0.003363636363636364 seconds.
--------------------------------------------
This is the result of top 95 hard problems:
Solved 95 problems.
Solved in 0.62 seconds.
Solved every problem in average 0.006526315789473684 seconds.
```

Using Sudoku Solver, we can solve a Sudoku Puzzle every time. If it is solvable, displaying a solution in the 9 * 9 grid. If it is not solvable, “Error!” message could remind the user there exists some errors.

6. Conclusion and discussion

In this work we showed that combining backtracking searching with constraints propagation could give an efficient solution for a 9 * 9 Sudoku puzzle. Backtracking searching has the advantage of giving a solution, but it is time consuming. Only using constraints propagation could not guarantee a solution but could help backtracking get less possibilities. So using backtracking searching with constraints propagation could provide us with an efficient method to solve Sudoku puzzle.

In the future, I could improve my work by implementing randomly generating a Sudoku puzzle and giving a solution, using other algorithm to solve Sudoku puzzle, and checking the difficulty of a Sudoku puzzle.

7. References

[1] Takayuki YATO, Takahiro SETA : “Complexity and Completeness of Finding Another Solution and Its Application to Puzzles”, 2013-10-20.
[2] Bertram Felgenhauer, Frazer Jarvis : “Mathematics of Sudoku I”, 2006-01-25
[3] Stuart J. Russell, Peter Norvig : Artificial Intelligence A Modern Approach (3rd Edition), ISBN-13: 978-0-13-604259-4.
[4] Gary McGuire, Bastian Tugemann, Filles Civario : “There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration”, 2013-08-31
[5] Peter Norvig : “Solving Every Sudoku Puzzle”
[6] David Eppstein : “Nonrepetitive Paths and Cycles in Graphs with Application to Sudoku”, 2005-07-20
[7] Allan Caine, Robin Cohen : “MITS : A Mixed-Initiative Intelligent Tutoring System for Sudoku”, Online ISBN : 978-3-540-34630-2 (2006)
[8] Miguel Nicolau, Conor Ryan : “Solving Sudoku with the GAuGE System”, Online ISBN : 978-3-540-33144-5 (2006)
[9] Wikipedia : “Sudoku solving algorithms”
[10] Rhyd Lewis : “Metaheuristics can Solve Sudoku Puzzles.”, Online ISSN : 1572-9397 (2007)
[11] Helmut Simonis : “Sudoku as a Constraint Problem” (2005)
[12] Donald E. Knuth : “Dancing Links”, 2000-11-15