PROJECT REPORT

CSE 537: Project 3

GAME SEARCH-Sudoku

Submitted by:

Shubhi Rani-110455118

Aakriti Mittal- 110562375

Introduction

Sudoku is a number placement puzzle. In this puzzle you are given an N x N grid of cells. The grid itself is composed of M x K sub-grids. You can place a single digit, drawn from 1 to N, in any cell. Initially the grid will have some of its cells partially filled. The objective of the puzzle is to complete the grid so that:

1. Every cell contains a digit.
2. No digit appears twice in any row, column of the N x N grid or in any row, column of any of the M x K sub-grid.

Figure 1 below is a 9 x 9 Sudoku puzzle made up of 3 x 3 sub-grids and Figure 2 is the solution to this puzzle.

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

Fig 1

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

Fig 2

We have modelled Sudoku as a CSP problem and have solved it using the following five CSP algorithms:

1. Backtracking
2. Backtracking + MRV heuristic
3. Backtracking + MRV + Forward Checking
4. Backtracking + MRV + Constraint Propagation
5. Min-conflicts Heuristic

Next section has Implementation Details and the last section contains results and Conclusion.

Implementation

A class `sudokuGame()` has been implemented which consists of variables and member functions which serve as helpers for our code.
Rest of the code handles the above mentioned algorithm as follows:

1. **Backtracking:**
    - This algorithm is implemented in csp.py file as :
      `def backtracking(filename):`
    - This algorithm is calling a recursive function which returns the solution by backtracking in the chronological order.

2. **Backtracking + MRV Heuristic:**
    - A better heuristic MRV has been used in addition to backtracking.
    - A domain of possible numbers at a particular cell is maintained beforehand and instead of starting from any arbitrary value (as in the case of backtracking), the starting point is taken as the cell whose domain consists of least number of values.
    - It has been implemented as `def backtrackingMRV(filename)in csp.py`

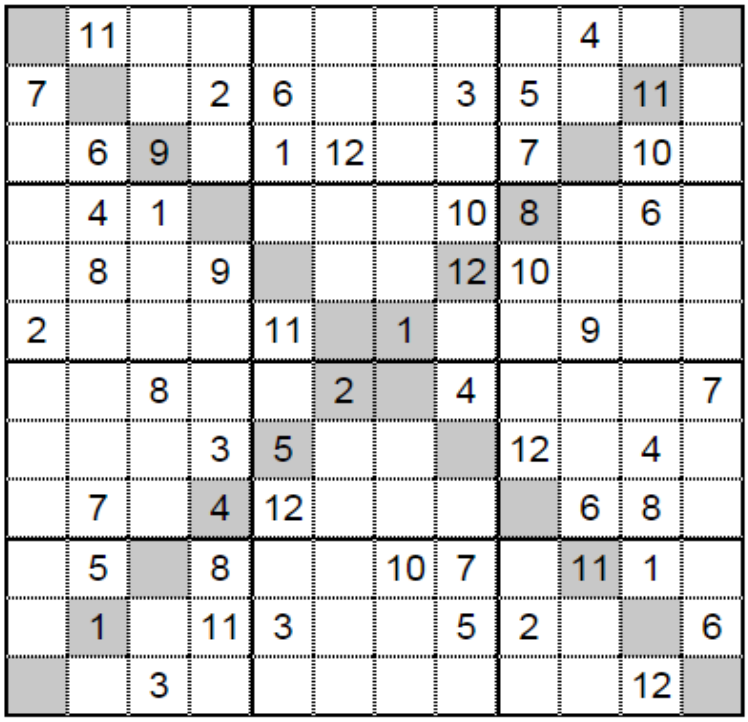
3. **Backtracking + MRV Heuristic+ Forward Checking:**
    - Forward checking has been used to construct the domains of the cells of Sudoku so that values can be selected from within that domain to save unnecessary computation.
    - It helps MRV in better computation.

4. **Backtracking + MRV + Constraint Propagation:**
    - Constraint Propagation has been added to MRV heuristic.
    - It checks arc consistency between every two cells and propagates that throughout the Sudoku matrix.
    - It has been implemented as def `backtrackingMRVcp(filename): in csp.py`

5. **Min-conflicts Heuristic:**
    - This heuristic first initializes the board with a particular assignment with broken constraints.
    - After that, any random variable is selected and corresponding to it, all constraints are satisfied.
    - The value for that random value is chosen by using min-conflict heuristic i.e which collides with least number of other values.
    - It is implemented as def `minConflict(filename):`

Results

The sample input Board Configuration is as follows:



The Result and analysis for various algorithms:

1. **Backtracking:**

   Backtracking:

   Execution Time: 28.1594936933

   Consistency Checks: 404745

   Solution: Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11], [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12], [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7], [1, 9, 11, 3, 5, 7, 8, 6, 12, 10, 4, 2], [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5], [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4], [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6], [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]

2. **Backtracking + MRV Heuristic:**

   backtrackingMRV:

   Execution Time: 7.58751232815

   Consistency Checks: 1507

   Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11], [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12], [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7], [1, 9, 11, 3, 5, 7, 8, 6, 12,

10, 4, 2], [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5], [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4], [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6], [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]

3. Backtracking + MRV Heuristic+ Forward Checking:

backtrackingMRVfwd:

Execution Time: 8.79178462294

Consistency Checks: 1407

Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11], [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12], [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7], [1, 9, 11, 3, 5, 7, 8, 6, 12, 10, 4, 2], [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5], [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4], [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6], [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]

4. Backtracking + MRV + Constraint Propagation:

backtrackingMRVcp:

Execution Time: 38.4720808372

Consistency Checks: 419

Solution: [[8, 11, 12, 1, 7, 10, 5, 2, 6, 4, 3, 9], [7, 10, 4, 2, 6, 8, 9, 3, 5, 12, 11, 1], [3, 6, 9, 5, 1, 12, 4, 11, 7, 2, 10, 8], [11, 4, 1, 12, 9, 5, 2, 10, 8, 7, 6, 3], [6, 8, 5, 9, 4, 3, 7, 12, 10, 1, 2, 11], [2, 3, 7, 10, 11, 6, 1, 8, 4, 9, 5, 12], [5, 12, 8, 6, 10, 2, 11, 4, 1, 3, 9, 7], [1, 9, 11, 3, 5, 7, 8, 6, 12, 10, 4, 2], [10, 7, 2, 4, 12, 1, 3, 9, 11, 6, 8, 5], [12, 5, 6, 8, 2, 9, 10, 7, 3, 11, 1, 4], [9, 1, 10, 11, 3, 4, 12, 5, 2, 8, 7, 6], [4, 2, 3, 7, 8, 11, 6, 1, 9, 5, 12, 10]]

5. Min-conflicts Heuristic:

The sample board configuration for this heuristic is as follows:

minConflict:

Execution Time: 3.799285

Consistency Checks: 4340

Solution: [[5, 1, 9, 7, 4, 8, 6, 3, 2], [7, 8, 3, 6, 5, 2, 4, 1, 9], [4, 2, 6, 1, 3, 9, 8, 7, 5], [3, 5, 7, 9, 8, 6, 2, 4, 1], [2, 6, 4, 3, 1, 7, 5, 9, 8], [1, 9, 8, 5, 2, 4, 3, 6, 7], [9, 7, 5, 8, 6, 3, 1, 2, 4], [8, 3, 2, 4, 9, 1, 7, 5, 6], [6, 4, 1, 2, 7, 5, 9, 8, 3]]

CSP Methods	**Execution Time**	**Consistency Checks**	**Comments**
Backtracking	28.15	404745	High Execution Time as it backtracks on the basis of chronological order without taking in consideration any heuristic. Highest number of consistency checks.
Backtracking + MRV	7.58	1507	Running best for 12*12 Sudoku as instead of starting with any random value, starting with the cell containing minimum number of possibilities. Number of consistency checks less than Backtracking as some states are getting pruned.
Backtracking + MRV + Forward Checking	8.79	1407	Execution Time is little higher than B+MRV as more time is taken in initial computation. However, consistency checks are reduced.
Backtracking + MRV + Constraint Propagation	38.47	419	Execution is highest as it computes the AC-3 dependencies for every blank cell with its conflicting cells. It takes more time computing values in as compared to search time for Sudoku. Consistency checks are reduced considerably because of pruning of states using constraint propagation.
Min-conflicts (9*9 Sudoku Board)	3.79	4340	Working till n=9. Min conflicts heuristic is NOT OPTIMAL for solving Sudoku as it has to compute and swap values for a large number of variable (cells). It is better for solving problems such as N-queens etc. which have a small number of variables.

References

- http://www.sudokuspoiler.net/SudokuX/SudokuX12
- https://en.wikipedia.org/wiki/Minconflicts_algorithm
- https://www.youtube.com/watch?v=QI0diwmx3OY