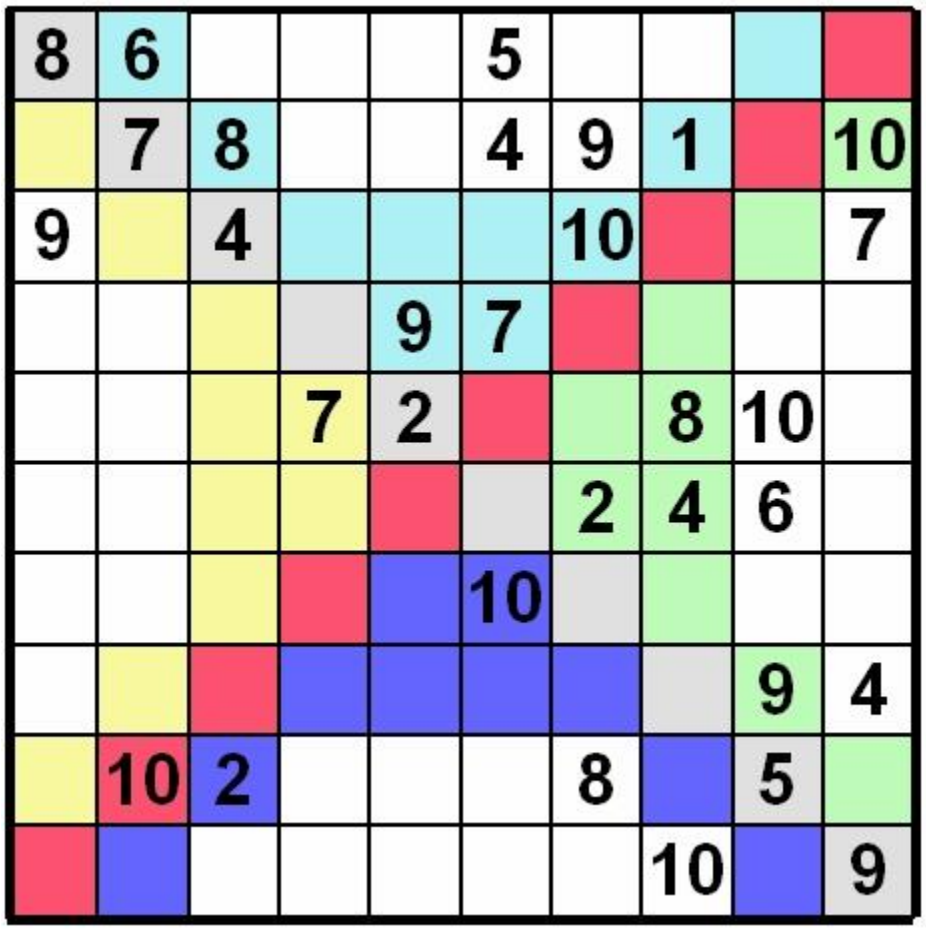
# PROJECT REPORT

**CSE-537 | ASSIGNMENT-3**




**Alpit Gupta (SBU ID: 110451714) | Vinayak Mittal (SBU ID: 110385943)**

**Artificial Intelligence | October 25, 2015**

# Introduction

**Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill N x N grid of cells. The grid itself is composed of M x K sub-grids. One can place a single digit, drawn from 1 to N, in any cell. Initially the grids 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 2(a) below is a 12 x 12 Sudoku puzzle made up of 3 x 4 sub-grids and Figure 2(b) is the solution to this puzzle.**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 11 | | | | | | | | 4 | | |
| 7 | | | 2 | 6 | | | 3 | 5 | | 11 | |
| | 6 | 9 | | 1 | 12 | | | 7 | | 10 | |
| | 4 | 1 | | | | | 10 | 8 | | 6 | |
| | 8 | | 9 | | | | 12 | 10 | | | |
| 2 | | | | 11 | | 1 | | | 9 | | |
| | | 8 | | | 2 | | 4 | | | | 7 |
| | | | 3 | 5 | | | | 12 | | 4 | |
| | 7 | | 4 | 12 | | | | | 6 | 8 | |
| | 5 | | 8 | | | 10 | 7 | | 11 | 1 | |
| | 1 | | 11 | 3 | | | 5 | 2 | | | 6 |
| | | 3 | | | | | | | | 12 | |

Figure 2(a)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

Figure 2(b)

**This project has implemented the following search techniques and provide a running and comparative analysis in detail:**

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

## Backtracking

The most obvious way to solve a Sudoku puzzle is to just pick the first empty square and assign one to that square. If the one conflicts with another value, then change it to two. Keep doing this until a value that does not conflict is found. Once a value that does not conflict has been found, pick another square and repeat this process. If a square has no possible values, then return to the previously assigned square and change its value. This method of search is known as a backtracking search. It is guaranteed to find a solution if there is one, simply because it will eventually try every possible number in every possible location. This algorithm is very effective for size two puzzles. But as the size of puzzle grows, the search space grows exponentially. Hence, it becomes difficult to solve large grid size puzzles.

**Output:**

***Execution Time: 8.2103485467***
***Consistency Checks: 404745***

***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]]**

## Backtracking + MRV heuristic

The minimum remaining values heuristic is used to alter order in which squares are guessed in order to reduce the number of branches at each level.
Basically instead of choosing the first empty square, the square with the least number of possible values is chosen. Once such a variable is picked, it assigns a value and starts the process again to find MRV for remaining variables. MRV is a great improvement over simple backtracking as rather than assigning values to every variable, it first tries to assign value to a variable which has least values remaining. Therefore, if this chosen variable doesn't hold true in further steps, then failure can be detected early and new value would be checked after backtracking.

**Output:**

***Execution Time: 2.77984790907***
***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]]**

## Backtracking + MRV + Forward Checking

In the earlier implementation of MRV + backtracking, situations like null domain set for child variables when a value is assigned to a variable is not checked. Thus, more backtracking steps are required to come to a stage where it would find a variable which has no legal value left. However, such scenarios can be easily avoided using forward checking. In this search implementation, as soon as a value is assigned to a variable, it first checks whether any variable sharing same row, column or grid has zero legal values left. If any such variable is found, then it tries to assign another value and hence more backtracking steps are saved.

**Output:**

***Execution Time: 3.31156958489***
***Consistency Checks: 1417***
***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]]**

## Backtracking + MRV + Constraint Propagation

Forward checking can only catch conflicts right before they cause a certain branch to fail. It is possible to detect errors even earlier and prune off entire branches. Once a value is assigned to a variable, it finds the legal remaining values of all its children. If any child has zero remaining values, then it immediately returns an error. However, if immediate children are having no conflict, then every child is checked for its legal values to find if it conflicts with its children. In effect, it checks children of children and thus saves lot of backtracking efforts.

**Output:**

***Execution Time: 15.5675608 06***
***Consistency Checks: 1257***
***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]]**

## Min-conflicts Heuristic

Min-Conflict Heuristic is a repair solution to Sudoku puzzle. First, it parses the input puzzle and for every vacant cell it randomly provides a value in the range of [1, N]. The input also provides total iterative counts for which the program runs. The algorithm finds an arbitrary constraint cell and finds minimum constrained value for this variable. This algorithm either gives success if a winning state is found within iterative count limit or else returns failure.

This heuristic is good for solving small grid puzzles or puzzles which already have a lot of values filled as input. However, for large grid sizes if there are more vacant positions, then the processing tends to infinity.

After iterating for 2, 00,000 times, the following output is generated with this algorithm:

**Output:**

***Execution Time: 154.11628558***
***Consistency Checks: 200000***

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

## Conclusion

Backtracking is good for puzzle with 2 x 2 grid size, where the search space is relatively very small. However, if we go for bigger grid size then its branching factor will increase and will take exponential time to solve the puzzle. Backtracking + MRV is a slight improvement over simple backtracking as it reduces the search space by choosing the variable which has remaining legal values left. This search is still good for short grid size Sudoku and would be able to solve in lesser time than doing only backtracking. Backtracking + MRV + forward checking is capable of solving 3 x 3 puzzle size with efficiency as it is able to detect failure early if any child of assigned variable has no legit values left in its domain space. Backtracking + MRV + constraint propagation employs more smart search technique by checking if any assigned variable leaves any of its child with values which are not consistent in advance. Hence, this technique helps in detecting failures in advance. Min-conflict heuristic would also be able to solve small grid size puzzles only.