**1. Introduction**

The report discusses about the designing of an intelligent algorithm to solve Sudoku puzzles of any grid size, and the development of the solution. It introduces the Sudoku puzzle, talks about the complexity and gives the solution. The solution addresses the solving of Sudoku puzzles rationally, with limited memory and processing power within reasonable time limits.

In addition, it illustrates the use of AI techniques in handling the similar problems. It further explains a methodology by taking the problem as a Constraint Satisfaction Problem and provides detailed systematic solution. In addition, the possible improvements of the solution are also discussed at the end.

**2. Problem**

*2.1 What is Sudoku*

• Sudoku is a simple game with a single rule: “Complete the grid by placing digits 1 through 9 ineach cell in such a way that only one of each digit is present in every row, column and box”.

• This rule is often accompanied by the following statement: “Each puzzle has a unique solution,which can be found by logic alone.”

*2.2 Brief History of Sudoku [1]*

The basis of Sudoku is the “Latin Square”, which has been known for thousands of years, but regained popularity when the Swiss mathematician Leonhard Euler made a study of them. Since it was first published by Dell Magazines in 1979, the 3x3-box constraint has become universally adopted and welcomed as a puzzle. The standard Sudoku consists of a 9x9 grid, with 81 squares. Different sized grids exist in variations of Sudoku.

*2.3 Solving*

Solving basic employs the method of checking all the possible candidates for given cell without violating

the rule - in each cell in such a way that only one of each digit is present in peers i.e. every row, column

and box. However, there are many advanced solving methods available generally viewed as pattern matching.

The given solution uses few of them to tackle the problem thus eliminate some of the possible

candidates selected

*2.4 Terminology [2]*

This introduces to Sudoku terminology with referring to a 9x9 grid. Shown in the picture

• Cell: Each position in the grid is called a “cell” or “square”.

• Row: The grid contains 9 “rows”. Each row contains 9 cells.

• Column: The grid can also be divided into 9 “columns”, each containing 9 cells.

• Boxes : A third method to divide the grid is in 3x3 “boxes”, each containing 9 cells

• Givens: A number of digits are already in place. These are the “givens”.

• Minimal Puzzles: A valid Sudoku has a unique solution.

• Peers: There are 9 cells in a house. Every cell is related to the 8 other cells in that house. A cell is a member of 3 houses: a row, a column and a box. 2 cells in the box are also a member of the same row, and 2 cells are also a member of the same column, leaving 20 cells in total that share one or more houses with any other cell.

Few other terminologies including Naked single, Naked double, Naked triple, Hidden Single,

Hidden double, Hidden triple, Full house, Squeezing, Cross hatching, Locked candidates and XWing

are used in the article but omitted here as they are used with less frequency in the article,

however, in case, could be consulted from online Sudoku repositories [2].

**3. Problem Review and references**

This problem is classified as Constraint Satisfaction Problem (CSP) and is comparable with coloring USA problem [3] in arc complexities. Coloring USA would take about one second to be solved in a normal computer with CSP with Minimum Remaining Value (MRV) and the complexity of Sudoku 9x9 puzzle is greater. The complexity could be emphasized by stressing the facts that there are 81 locations comparable to 50 in previous, user’s initial input, which demands a unique solution, and arc complexity is less than 10 in coloring USA but 20 in the latter. However, by introducing digits to represent the candidates and using logics to apply more constraints this could be solved faster, with less memory.

The Problem could be solved by creating all possible models (of higher orders) and just implement a simple search that would lead to an inefficient design [2]. Application of CSP type solution structures the problem solving and provides the basic skeleton. Further backtracking guarantees the completeness of the when it is solvable. However plain backtracking is comparable to uninformed search [3].

However, by selecting the candidate to be filled with MRV, we could speed up the process, by making sure that we are on the right track and if we are not reduce the time to find out it and backtrack. Forward checking (FC) could be used to increase the performance. However, it is only partially used in the solution to make the code as simple as possible small.

Therefore, to make the design more efficient we used CSP with Backtracking and improved its performance by and MRV with FC. Further algorithm tackles any grid size (limited by the platform word length though due to the use of bit level storage in integers) and for 25x25 the complexity is about 1000 times greater than Coloring USA problem.

**4. Design of the Solution**

*4.1 Program flow*

As far as the solution is considered,

• The user is able to define the size of the puzzle, which he desires to solve (9x9, 25x 25 etc).

• An array is created (Allocation of memory) to realize the puzzle, where each cell in the array

would indicate a single cage of the Sudoku puzzle.

*4.1.1 Initializing the system*

Initial values are obtained from the user or valid values are assumed for partially completed puzzles. The memory is allocated and necessary collection objects are created.

*4.1.2 Formulation of the possible candidates*

After the problem is initialized (cells filled i.e. constraints are provided), the possible candidates for a particular cell (empty cell) are formulated. This is kept in track by assigning a bit value a candidate. For Example in a 9X9 puzzle, Assume that a cell is capable of having numbers 3, 6, 8. So what the program does is that it creates an array of 9 bits (i.e. using the last 9 bits of an integer) and turns the 3rd 6th and the 8th bit on, so the cell would have a particular value. Thus spatial requirements are reduced and bit level logic is used to analyze hence processing speed is increased as the entry boils down into a stream of bits.

This allocation is done row wise, column wise and box wise as well.( In this way we are formulating the constraints for a given cell- this is the logical way a person would try to solve a Sudoku problem manually)

*4.2 Recursive Backtracking*

We use the Recursive backtracking algorithm to find the solution

Through minimum degree heuristic, we identify an empty cell with the least number of constraints – MRV – minimum number of possible candidates. For each possible candidate for that cell we make the assignment.

When we do an assignment in the recursive backtracking method we need to have ways to

1. Remember the changes made in the system state variables (Due to the fact the assignments are made to original system as it is very costly to keep local variables inside a recursive function)

2. Remember the Candidates and the cells related to the candidate (To undo when we unwind with recursive function)

For this purpose, we use a stack (another way of making things easy). We push all the changes made into a stack and pop when we unwind.

When it is found that path is wrong it is easy to undo the action by just a pop up assignment and unwind through Recursive backtracking and try another possible assignment for the cell (looping). Therefore, while continuing the recursive backtracking algorithm, when a failure is achieved the assignments are undone within the function and another possible path is chosen.

We keep the possible candidate negated, for our program identifies the level it needs to undo the previous action (pop the value out from the stack) until a negative number is reached. Because we use stack the undo action occurs in LIFO manner, which is mostly desired in this case.

*4.3 Minimum degree heuristic*

This includes CSP with MRV and forward checking algorithm and used to choose the next candidate to try assignments. The cell with minimum number of candidates is selected. When the puzzle is completed, i.e. no unfilled cell is found it returns success.

**5. Implementation of the Solution**

The solution is implemented in C++. The code can be easily converted to C, nonetheless C++ was chosento implement stack collection object. The source code of the main file can be found in Annexure A. Further, a web version of the solution is implemented in JavaScript and developed as a module for the popular CMS – Drupal. The source code is attached. Online version can be tried at

Try sudoku here

*5.1 User interaction*

1. If the user desires to create his own Sudoku problem he can assign the numbers, but this needsto be valid to produce a unique solution at the end.

2. If the user does not assign the numbers (or insufficient numbers), the program can generate a puzzle of all possible solutions and that may not be unique. However, in such case, solution is not guaranteed as the algorithm may fail in some loopholes.

*5.2 In implementing the solution following are taken into consideration and address in the following ways*

1. Memory consumption:

a. Use of a single integer to store all possible candidates by turning on the respective bits. I.e. bit level information storing, hence memory wastage is reduced. It is calculated to need less than 5KB in worst case i.e. 25 x 25 problem.

b. Use of stack of type integer to store changes (to undo if fails), hence memory wastage by allocating memory in prior is avoided.

2. Processing power optimization

a. Inline functions are used to generate the row, column or box id to find the candidates in the same house.

b. Number of possible candidates is stored in an array and incremented or decremented to prevent repeated calculations of them.

c. Instead of generating POSSIBLE CANDIDATES for a cell repeatedly, they are generated once at the beginning, stored in an array and updated dynamically throughout the flow of the program.

**6. Evaluations of results and ideas for improvement**

*6.1 Results*

The solution is tested with the puzzles attached in Annexure C and the times taken to solve them are given below according to their categories defined by Sudoku

• A < 1 ms

• B < 1 ms

• C < 1 ms

• D < 1 ms

*6.2 Improvements*

1. The solution overlooks the logic in selecting the heuristic. Instead of minimum degree heuristic,the following advance methodologies could be used to select the cell to be filled.

a. Using human like solving and utilizing Sudoku strategies like Naked single, Naked double, Naked triple, Hidden Single, Hidden double, Hidden triple, Full house, Squeezing, Cross hatching, Locked candidates, X-Wing, etc. This is possible to implement

b. Through machine learning and patter matching for a given pattern of filled candidates, the next cell to be filled can be selected. This can be barely put into code due to the randomness and infinite number of patters possible.

2. The aforementioned strategies can also be used to pick the candidate out of all possible candidates for an unfilled cell. Currently they are selected in ascending order of their values.

3. The solution uses a single integer to store all the candidates, by turning on the respective bits, thus limited in grid size by the word length of the platform which is usually of 32 bits, which makes the maximum grid size 25 x 25 (25 < 32). This limitation could be overcome by using virtualization through implementing this through a class for which logic operators (&, |, ~) used in the solution could be overloaded and dynamic length is allowed. This is possible in C++.

4. For the desktop version, a GUI can be provided.

**7. References**

[1] History of Sudoku: Roots and Development of Sudoku, http://www.sudokutips.com/about\_sudoku.php , last accessed JULY 2007.

[2] Sudoku Guide: http://www.sudocue.net/guide.php , last accessed JULY 2007.

[3] Artificial Intelligence A Modern Approach, Staurt Russel and Peter Norvig, Pearson Education INC, 2003.

[4] On magic squares http://arxiv.org/abs/math.CO/0408230 , last accessed JULY 2007.

[5] Enumerating possible Sudoku grids http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf , last accessed JULY 2007.

[6] Sudoku enumeration problems http://www.afjarvis.staff.shef.ac.uk/sudoku , last accessed JULY 2007.

[7] Sudoku Maths http://www.sudoku.com/boards/viewtopic.php?t=44&postdays=0&postorder=asc... , last accessed JULY 2007.

**Notes on Files attached**

C++ code is written for and can be compiled with VS 6.0.

Jquery module is written in Javascript with Jquery for drupal cms.

The code can be extracted from jquery.sudoku.js if is to be used elsewhere.