Solving Classic Sudoku Puzzles using Crook's Algorithm

Imee Compra, Sophia Dalumpines, Christine delos Reyes, Lea Flor
College of Science, Technological University of the Philippines - Manila
imee.compra@tup.edu.ph
sophia.dalumpines@tup.edu.ph
christine.delosreyes@tup.edu.ph
lea.flor@tup.edu.ph

Abstract – Solving a Sudoku puzzle can be done in many ways, including the Backtracking method. This method involves trying out numbers and, if it finds that a particular number does not lead to a solution, it backtracks and goes back to the previous step to start over. To prevent infinite loops, most backtracking algorithms have a stopping condition. The algorithm stops when either: 1) all empty spaces have been checked but the solution is not found, or 2) the board is filled without any empty spaces. Another method is using Crook's Algorithm has a set of steps that include Markup, Singleton discovery, locating Preemptive Sets, and removing the possible numbers that are part of these sets.

This paper will show the steps and approaches to solving a Sudoku puzzle using Crook's Model [1].

Keywords: *Sudoku Puzzle, Backtracking, Crook's Algorithm, Preemptive Sets*

I. INTRODUCTION

Sudoku puzzles are a popular form of entertainment and challenge for millions of people around the world. Sudoku is played on a 9x9 board with 81 cells, divided into nine non-overlapping 3x3 sub-boards. James Crook created an algorithm which is "A Pencil-and-Paper Algorithm" and is commonly known as Crook's Algorithm that solves Sudoku Puzzles. The algorithm follows all the rules of Sudoku and employs a more mathematical approach to solving the

puzzle. To solve a Sudoku puzzle, every row, column, and the box must contain the numbers 1-9 without any duplicates. To make solving easier, possible numbers for each cell are written down, called the cell's markup. The primary tool for solving the puzzle is using preemptive sets, which are groups of potential numbers for certain cells, with the goal of either finding a solution or having to choose one of two or more numbers from the cell's markup. A preemptive set consists of numbers from 1-9 and is a group of 2 to 9 numbers that have the potential to occupy a specific set of cells exclusively. The range of the set is determined by the location of its cells in a row, column, or box. When the size of the set is 2 or 3, its scope can be a combination of a row and box, or a combination of a column and box [2].

II. RELATED WORK

A Latin square of rank n is a square array with dimensions n x n that contains numbers from 1 to n such that each row and column has all the numbers from 1 to n. Sudoku puzzles are a type of Latin square with rank 9, but not all Latin squares of rank 9 are Sudoku puzzles as there is a condition that must be met for the nine 3x3 sub-grids. A λ-coloring of a graph is a mapping of the vertices of the graph to numbers 1 to λ such that no two adjacent vertices have the same color. The minimum number of colors needed to properly color a graph is called the chromatic number of the graph and is represented by $\chi(G)$. In a 9x9 Sudoku puzzle, there are 81 vertices, each representing a cell in the grid. Two vertices are considered adjacent if their corresponding cells are in the same row, column, or sub-grid. Solving a Sudoku puzzle involves eliminating the possibilities of colors in each row, column, and sub-grid until the puzzle is completed [3].

A preemptive set is a set of numbers from 1 to 9 with a size ranging from 2 to 9, which can only occupy a set number of cells exclusively. This means that no other numbers from 1 to 9 can occupy those cells. The range of a preemptive set refers to a row, column, or box that contains all the cells occupied by the set. When the size of the set is 2 or 3, the range can be either a combination of row and box or column and box.

Theorem 1 (Occupancy Theorem): If a set X is a preemptive set in a Sudoku puzzle, then any number in X that appears in cells outside of X within its range cannot be part of the puzzle solution. This is because if

a number in X is chosen for a cell outside of X, the number of choices for the cells in X would be reduced to m-1, resulting in one unoccupied cell in X, which goes against the definition of a Sudoku solution. To maintain a partial solution, all numbers in X must be eliminated wherever they appear in cells outside of X within its range.

Theorem 2 (Preemptive Sets): There is always a preemptive set that can be used to reveal a hidden set, transforming it into a preemptive set, with the exception of singleton sets. Hidden sets, especially singletons, and pairs, are useful as they are easier to identify than the accompanying preemptive set [4].

III. EXPERIMENT

A. Data Collection

The researchers gathered different sudoku puzzles from puzzle books and online sudoku sites. The system accepts the puzzle input through an excel file.

4	8	0	0	0	0	5	0	6
0	0	0	0	8	0	0	0	3
3	0	2	0	0	0	0	9	0
2	0	0	0	0	7	0	0	0
0	0	9	0	0	0	0	0	1
0	0	0	6	0	0	4	0	7
0	9	0	0	0	5	0	1	8
0	0	5	1	0	0	0	0	0
6	0	0	3	0	0	0	0	0

Fig. 1 Sudoku Puzzle in an Excel File

Zeros in the excel input represents the empty cells in the sudoku puzzle. Only one number per excel cell and one puzzle per sheet.

B. Crook's Algorithm

In solving a sudoku puzzle, Crook's algorithm is only applied after the fundamental rules are performed and all the possible obvious answers are found.

Finding preemptive sets is a significant process in Crook's algorithm. Shown below are the snippet codes of how each row, column, and box are being checked for a possible solution.

```
def crook_gridScan(candidates, soln): #scan each 3x3 boxes using crook's algorithm
    for i in [1, 4, 7]:
        for j in [1, 4, 7]:
            markups = {key: value for key, value in candidates.items()
                       if key[0] in puzzleRange(i) and key[1] in puzzleRange(j) and len(value) > 0}
            crookAlgo(markups, candidates, soln)
    return 0
```

Fig. 2 Applying Crook's Algorithm on each box

```
def crook_columnScan(candidates, soln): #scan each column using crook's algorithm
    for j in range(1, 10):
        markups = {key: value for key, value in candidates.items()
                   if key[1] == j and len(value) > 0}
        crookAlgo(markups, candidates, soln)
    return 0
```

Fig. 3 Applying Crook's Algorithm on each column

```
def crook_rowScan(candidates, soln): #scan each row using crook's algorithm
    for i in range(1, 10):
        markups = {key: value for key, value in candidates.items()
                   if key[0] == i and len(value) > 0}
        crookAlgo(markups, candidates, soln)
    return 0
```

Fig. 4 Applying Crook's Algorithm on each row

```
def crookSol(candidates, soln): #solve using crook's algorithm until possible
    while True:
        prevSolution = copy.deepcopy(soln)
        crook_rowScan(candidates, soln)
        crook_columnScan(candidates, soln)
        crook_gridScan(candidates, soln)
        if soln == prevSolution:
            break
    return 0
```

Fig. 5 Applying Crook's Algorithm until no new solution is found

The application of sudoku's fundamental rules and Crook's Algorithm will be performed on a loop until the final solution to the puzzle is found.

C. *User Interface and Testing*

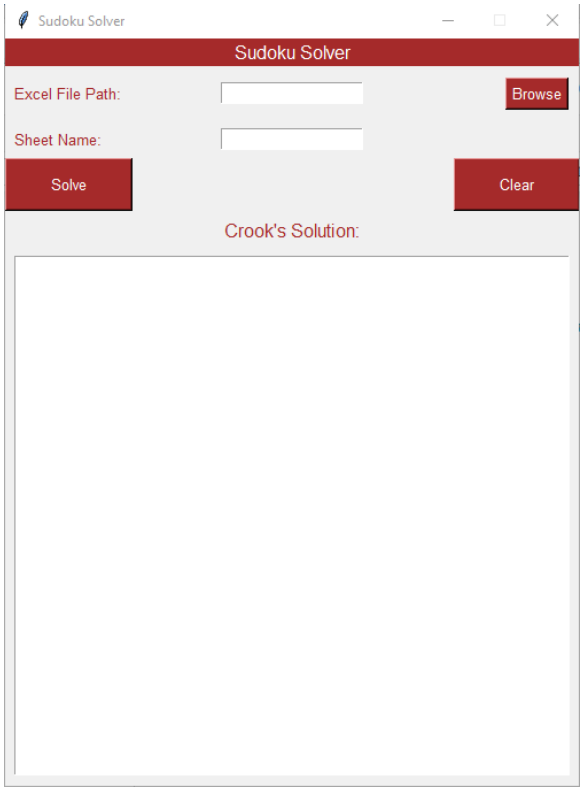

Fig. 6 User Interface of the Sudoku Solver

Shown above is how the user will be able to use the system to solve a sudoku puzzle. The user will be asked to put the excel file path of the puzzle using the browse button. The sheet name will be required next. After completing those two fields, the user can now click the solve button and the solution for their entered sudoku puzzle will be displayed in the text box. A clear button is also provided for the user to easily clear the interface output and start solving a new puzzle.

IV. RESULTS

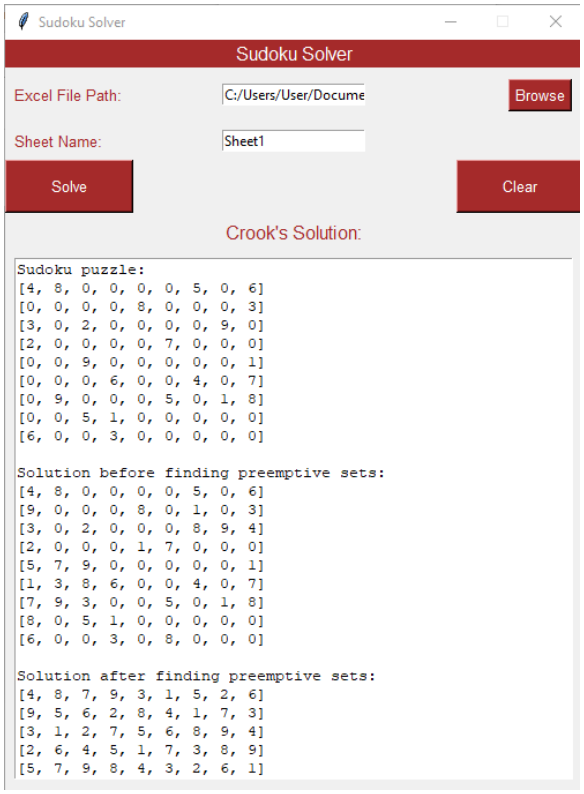

Fig. 7.1 Output of the Sudoku Solver

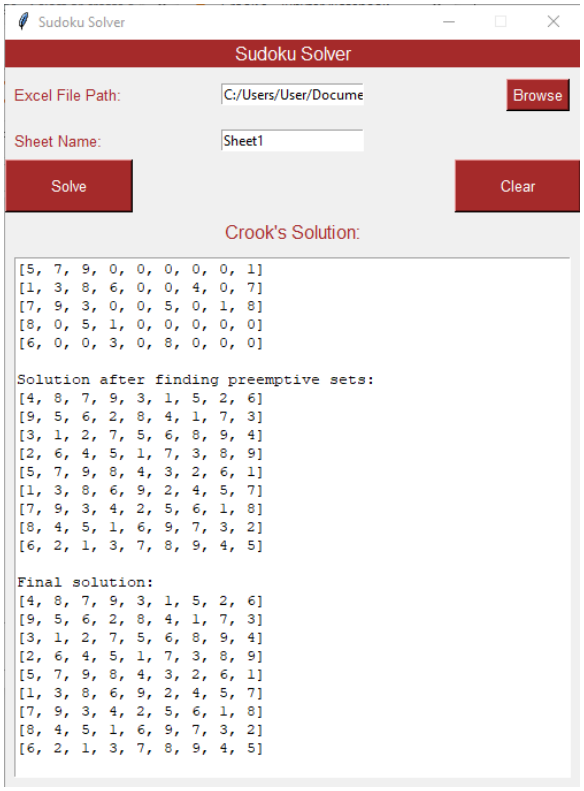

Fig. 7.2 Output of the Sudoku Solver

The images above show the step-by-step solution of a sudoku puzzle using Crook's Algorithm. Displayed first is the original puzzle. Then after is the initial solution after applying the basic sudoku rules and before applying the preemptive sets. Displayed after is the solution after finding the preemptive sets of the puzzle. And printed last, is the final solution found.

The system was tested for several puzzles with difficulties ranging from easy to hard. In testing the system for low-difficulty puzzles, the full solution is immediately found even before finding the preemptive sets. Furthermore, in testing high-difficulty puzzles, after finding the preemptive sets, the final solution is presently found.

V. CONCLUSION & RECOMMENDATION

The system was able to solve a given puzzle and was also able to show the then-current solutions before and after applying Crook's Algorithm. After testing different puzzles, the final solution is immediately found after applying Crook's algorithm. This shows how optimal Crook's algorithm is as a sudoku-solving technique.

The researchers advise future researchers to develop a system where the solver can be applied to other types of sudoku puzzles such as Irregular Sudoku, Killer Sudoku, and Samurai Sudoku.

VI. REFERENCES

[1] Jordan (2020). Solving Algorithms To Dominate Sudoku, Quick Review. https://www.valvetime.net/solving-algorithms-to-dominate-sudoku-quick-review/

[2] J. F. Crook (2009) A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles. Volume 56, Number 4

[3] Herzberg, Agnes & Murty, Ram. (2007). Sudoku squares and chromatic polynomials. Internationale Mathematische Nachrichten. 54.

[4] J.F. Crook (2009) A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles. Volume 56, Number 4