SUDOKU PUZZLE SOLVING BY USING AC-3 AND BACKTRACKING ALGORITHMS

Burcu Güney, **Kaan Berke Uğurlar** Istanbul Aydın University Software Engineering Department,Istanbul,Turkey, *guneyburcu@outlook.com*, *kaanberkeugurlar@gmail.com*

Abstract

Sudoku is a puzzle with basic and simple rules for artificial intelligence problems. In this paper, we will try to show which algorithm is faster for optimal results by using backtracking algorithm and AC-3 algorithm which is one of the arc consistency algorithms used in CSP solutions.

Keywords: Sudoku, Arc consistency (AC-3) algorithm, Backtracking Algorithm,Time, Perfomance

1. *Introduction*

Sudoku is a combinatorial puzzle in which numbers are be placed in a 9x9 grid, which is divided into 3x3 sub-grids. The grid is partially completed, and each Sudoku has a distinct solution. The objective of the puzzle is to fill the grid with numbers [1-9], without the repetition of a number in a line, column or the sub-grid. The numbers are used only once in each column and row, and the sum of each row and column must be equal. Difficulty of the problem depends on the partially completed grid.

The basis of the Sudoku puzzle is “Latin Squares”. It was developed by the Swiss mathematician in the 18th century. Some of the puzzles in newspaper was based on Latin Squares. The modern Sudoku puzzle used today was first published in 1979 in the USA by Howard Garns under the name "Number placement". Basically, it consists of 81, which is 9x9, boxes in total [1].

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

Figure 1: Sudoku Puzzle

AC-3 (Arc Consistency) Algorithm

The developed Arc algorithms consist of nodes and arcs between points. The first developed AC-1 algorithm controls the value that each node must receive and removes the inappropriate value. This algorithm is a fixed point algorithm. It proceeds by repeating each node and the suitability of the spring.

The developed version of this algorithm, AC-3 by Alan Mackworth in 1977, was considered optimal when it was first developed. Unlike AC-1, the AC-3 algorithm does not check each spring repeatedly, only checks the springs to that node and removes inappropriate values.

"The following figure shows these dependencies between pairs of values. If we remove the value from the domain of variable V2 (because it has no support in V1), we do not need to check values a, b, c from the domain of the V3 because they have all the other domain in V2. However, we have to remove the value from V3." [2]

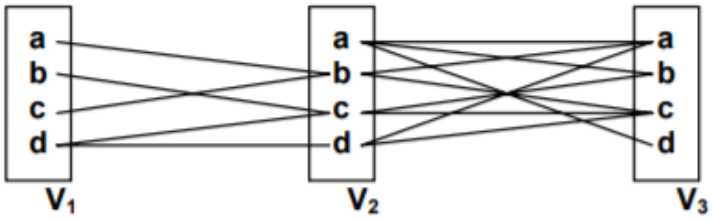

Figure 2: AC-3 Algorithm's Mechanism

Backtracking Algorithm

Backtracking algorithm was invented by D. H. Lehmer in the 50s and it is one of the heuristic search algorithms frequently used in artificial intelligence. It is often used especially for solving the Sudoku puzzle. This algorithm tries to solve the puzzle as recursive. Trying the values in the range of the mechanism of action on the grid, it returns to the beginning when it encounters an inappropriate value and tries the new value. So, in short, this algorithm attempts to re-establish the probability of returning to the last optimum value when it encounters any obstacle.

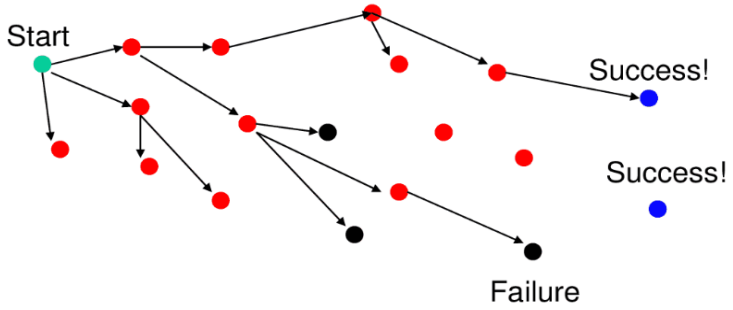

Figure 3: Backtracking Algorithm's Mechanism

We aim to evaluate both algorithms in terms of time and performance by using the top 500 data of the publicly shared Kaggle's 1 million Sudoku puzzles dataset [3].

2. Related Work

Several experimental algorithms have been studied on the Sudoku puzzle. Many developers have solved the puzzle using hybrid algorithms or different heuristic, metaheuristic, genetic algorithm with various optimization algorithms (Ant, Bee etc.) and tried to find the most efficient result in terms of time, performance and minimum step. The best examples are Tabu and Cuckoo algorithms, and AC-3 [4] reported that they achieved much faster results when used together with hybrid.

The puzzle was solved by 5 different algorithms (Brute-Force, Simulated Annealing, Genetic algorithm, Harmony Search algorithm and Graph Referencing algorithm) by different developers [5]. Puzzles were solved, the optimum result in terms of time and performance Graph They have obtained the referencing algorithm.

When the articles of the studies are examined, the algorithms used in pure form showed lower performance in terms of time and performance than hybrid algorithms. Regardless of these studies, we try backtracking algorithm which is accepted as pure algorithm and AC-3 algorithm. Since we know that hybrid algorithms are much better, we don't have enough time to apply some algorithms as hybrids, so we will apply the main algorithms and evaluate the results.

3. Overview, Methods and Tools

The backtracking algorithm starts solving the sudoku puzzle as soon as it takes the string which consists of a sudoku puzzle. Controls the column and column at the same time. If the assigned value is correct, the cycle continues and proceeds to the next move. If the assigned value is not correct, it checks again and returns to the previous move and tries the new values. output.txt writes solutions line by line.

Pseudocode structure of backtracking algorithm:

1. Backtracking procedure declaration is made.
2. Sudoku puzzle and order of sudoku (N) is taken as an input.
3. Sudoku solving function which is named as ‘Solve’ is created.
4. If there is an empty cell then execute the lines below.
5. Declare Number variable to 1, execute the lines [6 - 14] and increase the Number variable by 1 per iteration as long as Numbers is not equal to N^2.
6. If Number is eligible to be in the place where it was put temporarily, execute the lines [7 - 11]
7. Place the Number variable permanently to the place.
8. Call the Solve function and if it returns True then execute the line below.
9. Declare a variable as Output and make it equal to ‘Sudoku Solved’.
10. End of the if condition on 8^{th} line.
11. Make the cell empty.
12. End of the if condition on 6^{th} line.
13. End of the for loop on 5^{th} line.
14. Return False
15. If the condition on the 4^{th} line is not true then execute the line below.
16. Declare a variable as Output and make it equal to ‘Sudoku Solved’.
17. End of the backtracking procedure.

```
1: procedure BACKTRACKING
2:     Input Sudoku Puzzle, Order of Sudoku (N)
3:     Function (Solve)
4:     if Find an empty cell then
5:       for Number = 1: N^2 do
6:         if Safe to fill the Number then
7:             Fill the empty cell with the number
8:             If (Solve) then
9:                 Output: Sudoku solved
10:              end if
11:            Empty the cell
12:        end if
13:      end for
14:      Return False
15:   else
16:       Output: Sudoku solved
17:    end if
18: end procedure
```

Figure 4: Pseudocode of Backtracking

The AC3 algorithm also starts solving the sudoku puzzle like backtracking algorithm does. The difference of AC3 algorithm from backtracking algorithm is that AC3 algorithm assigns an array which has domain numbers of the sudoku system to every empty place in the sudoku puzzle. Afterwards, algorithm detects every neighbor of the arrays and one by one removes the ineligible numbers from the domain arrays. At the very end of the program, solutions will be written into output.txt file.

Pseudocode structure of AC3 algorithm:

1. AC3 function declaration
   - 1.1. Checks if is there an inconsistent circumstance, returns False but if there is not then vice versa; returns True.
   - 1.2. Inputs are processed individually.
   - 1.3. Local variables, which are queue, initial domain and etc. , are created regularly.
   - 1.4. A while loop will be run as long as queue is not empty.
     - 1.4.1. Queue's first element will be removed and assigned to X_i and X_j variables.
     - 1.4.2. Revise function will be called by sending csp, X_i and X_j variables as parameters then if the function returns true, following steps will be executed.
       - 1.4.2.1. First program checks D_i variable's size whether it is equal to zero or not. If it is, then returns False.
       - 1.4.2.2. For each variable in (X_i.NEIGHBORS – X_j) variable will be signed as X_k.
         - 1.4.2.2.1. Add (X_k, X_i) tuple to the queue.

2. Return True
    2.1. Revise function declaration
    2.2. Return True if the domain of X_i is already revised.
    2.3. Declare a variable which is named as revised and make it equal to False.
    2.4. For each variable in D_i variable will be assigned to x variable.
        2.4.1. If no value y in D_j allows (x, y) to satisfy the constraint between X_i and X_j execute below.
            2.4.1.1. Remove x from D_i
            2.4.1.2. Make revised variable equal to True
        2.4.2. Return revised variable.

```
Function AC-3 (csp)
returns False if an inconsistency is found, True otherwise
inputs: csp, a binary CSP with components ( X, D, C )
local variables: queue, a queue of arcs, initially all the arcs in csp
while queue is not empty do
    (Xi, Xj ) = REMOVE-FIRST (queue)
    if REVISE (csp, Xi, Xj) then
        if size of Di  = 0 then return False
        for each Xk in X i NEIGHBORS – {Xj} do
            add (Xk , X i) to queue
return true

function REVISE (csp, Xi, Xj)
returns True if we revise domain of Xi
revised = False
for each x in Di do
    if no value y in Dj allows (x,y) to satisfy the constraint between Xi and Xj
then
    delete x from Di
    revised = True
return revised
```

Figure 5: Pseudocode of AC-3

This solver was developed with Python language in PyCharm Professional and Community IDE and on computers with Window 10 Home operating system, 8 Gb RAM and MacBook Pro OS operating system 16 Gb RAM.

4. *Architecture, Algorithms, Models and Data*

Initially, we should clarify PEAS for our agent.

Agent: Sudoku solver

Performance measure: Efficiency, accuracy

Environment: Sudoku board

Actuators: Screen display

Sensor: Input from file

Then implemented backtracking algorithm which ends when there are no more solutions to the first sub-problem to solve sudoku puzzles based on the PEAS which is declared above and this result better than habitual.

Lastly, AC3 algorithm that is the one most often taught and used in very simple constraint solvers was applied to sudoku problem and solved the vast majority of puzzles. Even though it tried its best, it was not able to solve as many puzzles as backtracking algorithm solved.

We use ‘1 Million Sudoku Puzzles’ dataset on Kaggle which is publicly available. Dataset has two columns; Quizzes and solutions. In the first column, there are one million sudoku puzzles which are strings with eighty-one length and in the second column there are one million strings as well but this time all of them are solved forms of the first column. AC3 algorithm is one of a series of algorithms used for the solution of constraint satisfaction problems (or CSP's). AC-3 operates on constraints, variables, and the variables' domains (scopes). A variable can take any of several discrete values; the set of values for a particular variable is known as its domain. A constraint is a relation that limits or constrains the values a variable may have. The constraint may involve the values of other variables.

Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution. 2 Architecture of the sudoku solver program that we coded consists of three classes; Main, AC3 and Backtracking classes. Main class gets strings of sudoku puzzles from a file and sends them to solve functions in AC3 and Backtracking classes as a parameter and takes returned strings that is solved forms of unsolved sudoku puzzles. Main class writes all the solution to a file if user want to.

Our project UML class diagram is below:

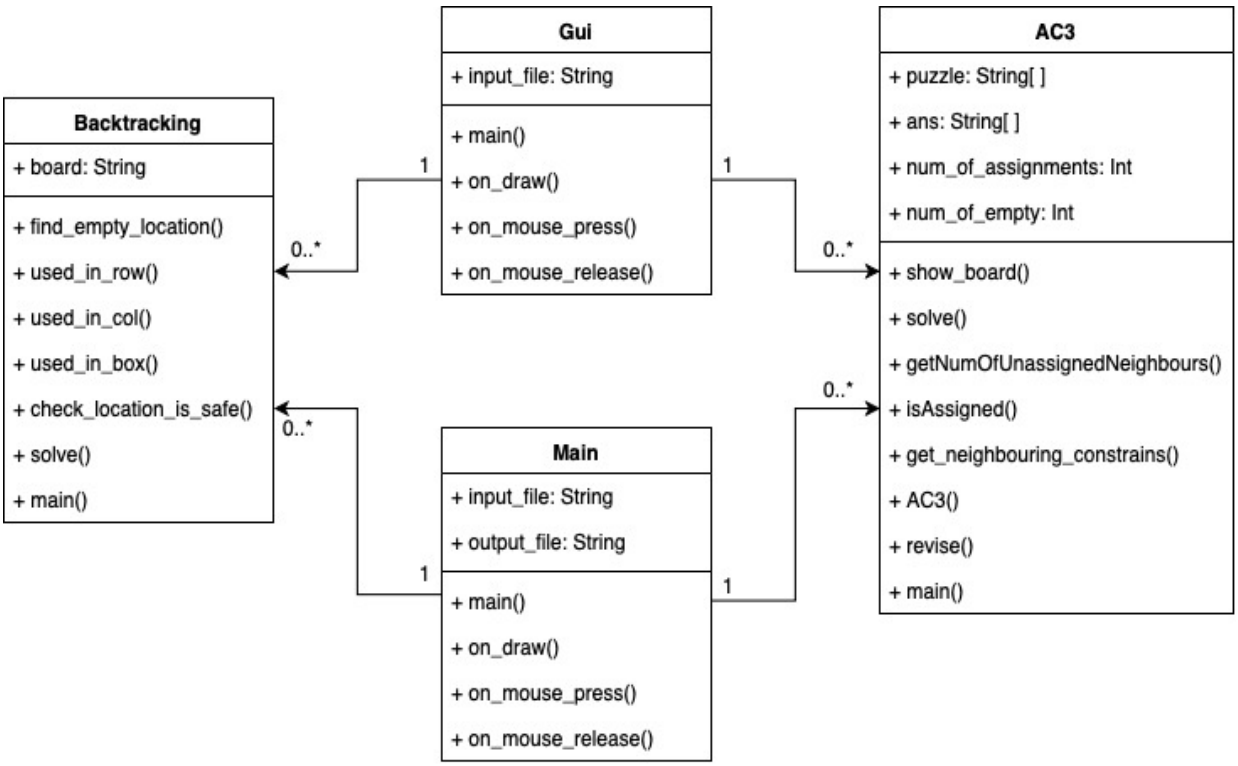

Figure 6: UML Class Diagram

4.1 Main.py

When you call main.py you need to send two parameters. One is input file and the other one is output file. Input file should have included one sudoku puzzle string, that is 81 length of, at a line. Main.py reads the input file and stores every sudoku puzzles into an array. Initially the program creates an instance of AC3 algorithm for each sudoku puzzle in the array and returns their solutions. Thereafter, the program creates an instance of Backtracking algorithm for each sudoku puzzle in the array and return their solutions. While these solving progresses is processed by the program, it also calculates the time that is taken by every algorithm which is applied. Finally, an output file which was given in the beginning of the process will be filled by solutions one by one and written into the file.

4.2 GUI.py

When gui program starts, pyglet draws the lines for sudoku board and puts buttons on the very right part of the scene. The program fills the empty boxes with numbers which is contained by sudoku puzzle strings in the input.txt file. There are two different buttons. First one is 'Reset' button that works for resetting the board and draw new sudoku puzzle. Second button is 'Start' button which starts the sudoku solving process and solves the sudoku that is on the board. As soon as 'Start' button is clicked, the process is triggered and creates an AC3 instance by sending current board to it as a parameter then AC3 returns its solved form. After AC3 solves the sudoku, or could not solved, backtracking instance would be created like AC3 was created.

Following, backtracking return the solution if it could solve it and at the end sudoku board will be filled with the solution. Solved numbers would be colored as green and on the right part of the scene there will be two different labels: 'AC3 Algorithm Taken Time' and 'Backtracking Algorithm Taken Time'

5. *Analysis, Experiments and Design*

The Backtracking and AC-3 algorithms were evaluated together and separately by the program. 500 various sudoku puzzles were solved by using both the algorithms and they generally took place in less than 0.5 seconds for 500 sudoku puzzles.

Backtracking algorithm was found to be behind in time and performance compared to AC-3 algorithm. Looking at the working mechanism of this algorithm, it returns to the beginning to find the right move after every incorrect move, and its restarting puts it behind AC-3. The average solution time is less than 0.3 seconds.

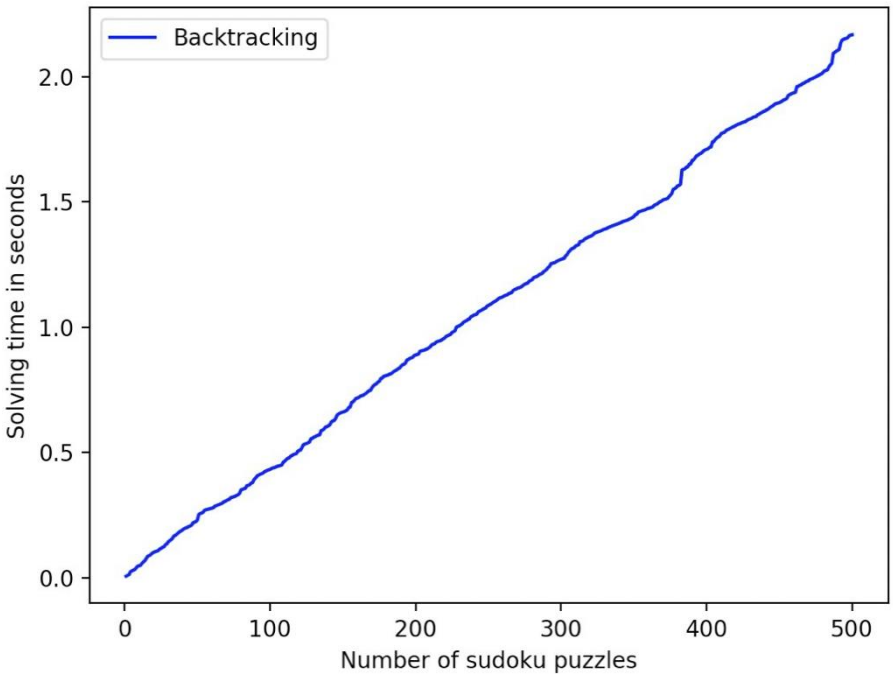

Figure 7-a: Number of sudoku puzzles over solving time in seconds based on backtracking algorithm.

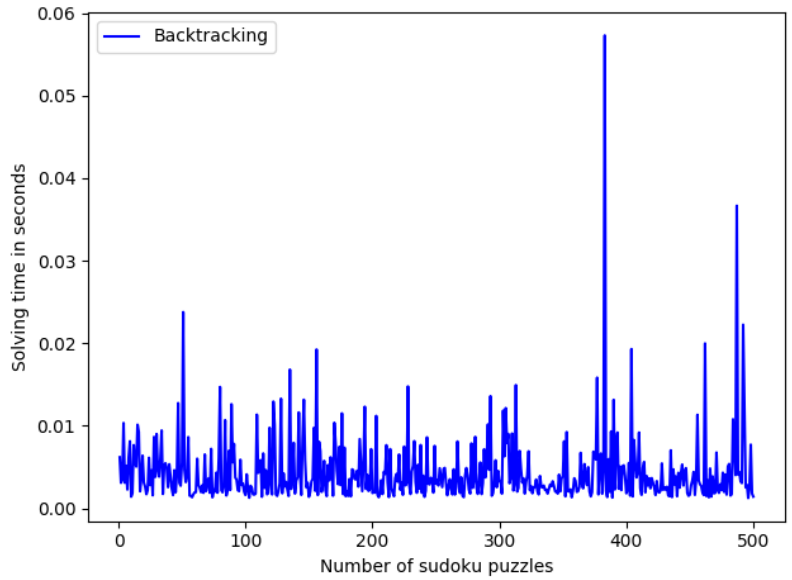

Figure 7-b: Solving time in seconds based on backtracking algorithm for each sudoku puzzle.

500 puzzles are solved with AC-3 algorithm. This algorithm is generally more successful in terms of time and performance than the backtracking algorithm. In the main grid and sub-grid solutions, it is faster in the solution because it only evaluates the possible values and returns to the previous value at each wrong value. The average solution time is 0.1-0.2 seconds.

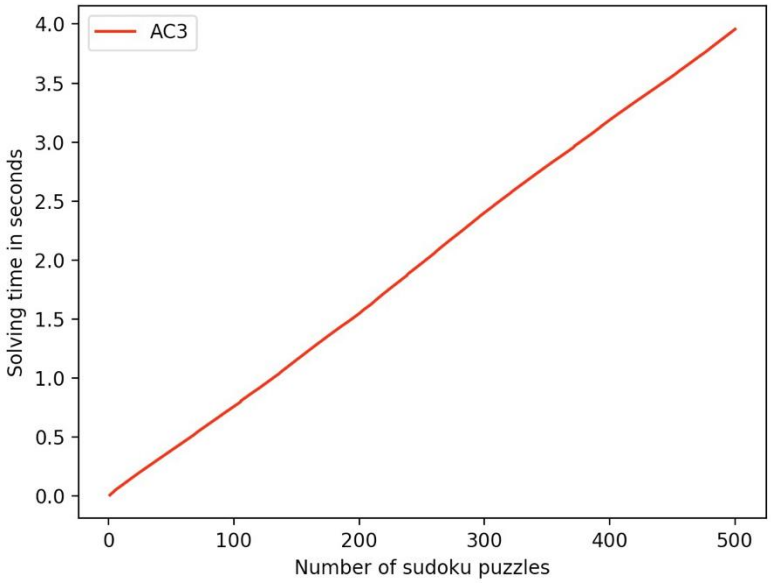

Figure 8-a: Number of sudoku puzzles over solving time in seconds based on AC3 algorithm

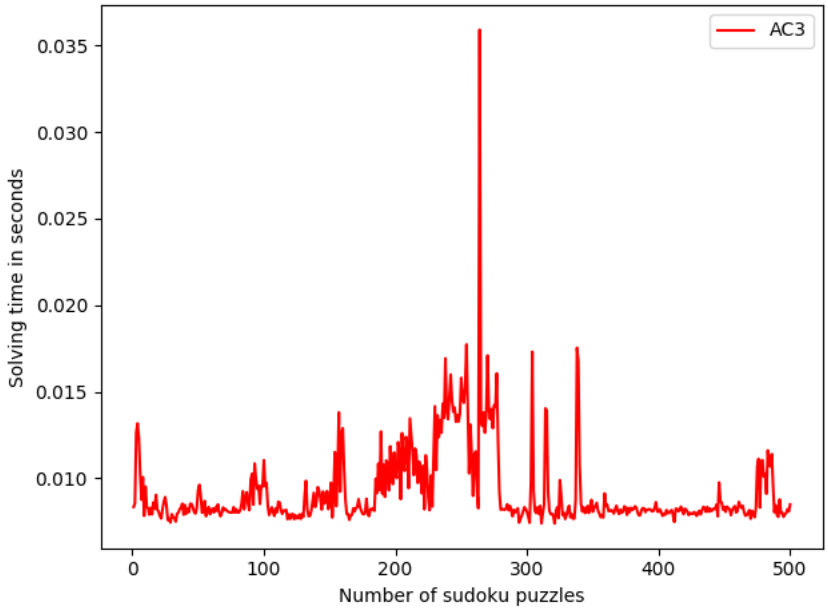

Figure 8-b: Solving time in seconds based on AC3 algorithm for each sudoku puzzle.

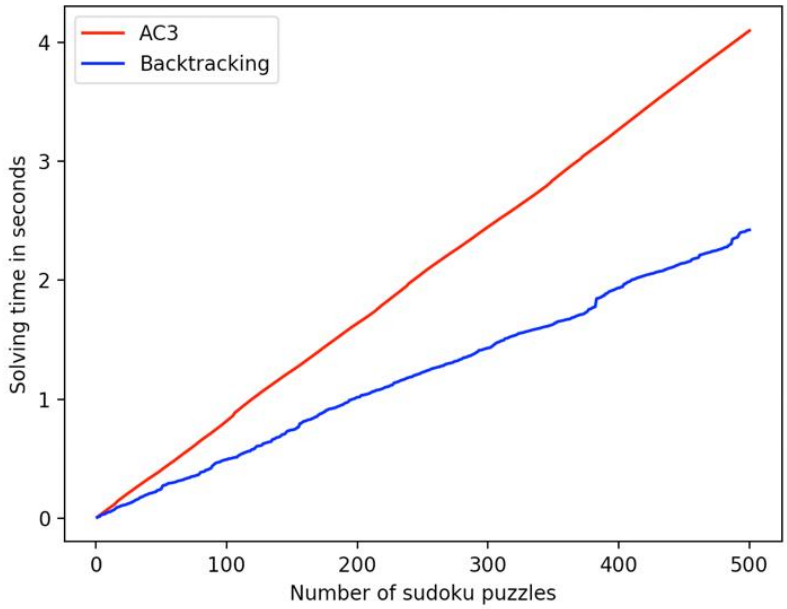

Figure 9-a: Comparison of backtracking and AC3 algorithms based on how many seconds for all sudoku puzzles.

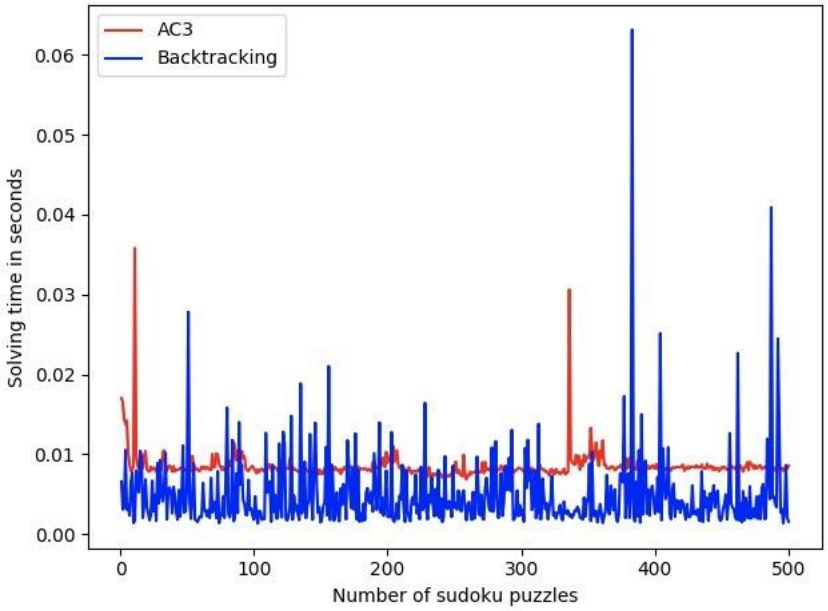

Figure 9-b: Comparison of backtracking and AC3 algorithms

based on how many seconds does one run take.

The main trouble we encountered was conflict between algorithm processes. Initially, we implement both algorithms and inspected the results then as far as we could checked all of them, the result was obvious; Backtracking algorithm was much better then AC3 algorithm. Moreover, we thought what if we apply these algorithms on one of the hardest sudoku puzzle. The result surprised us without a shadow of a doubt. AC3 algorithm was 160x faster than backtracking algorithm. We could solve this issue which is not a real problem because these two are very different algorithms and backtracking algorithm can be counted as a brute force algorithm because of it takes O(n)-1 time in theory.

6. *Conclusion*

In this article, we implemented sudoku puzzle backtracking and AC-3 algorithms using python language and compared the results. According to the results, the best time and performance to solve 500 puzzles in 9x9 main grid and 3x3 sub-grid size were applied with AC-3 algorithm. In the future, this involves developing the puzzle in hybrid form using optimization algorithms along with the algorithms we use for better performance and shorter time. Overall there is space for larger studies with more algorithms for Solving Sudoku Puzzles of other sizes.

REFERENCES

[1] The history of Sudoku, https://sudoku.com/how-to-play/the-history-of-sudoku/

[2] R.Bartàk ,"Constraint Propagation and Backtracking-Based Search",1995,pg 16, https://www.math.unipd.it/~frossi/cp-school/CPschool05notes.pdf

[3] 1 Million Sudoku Games, https://www.kaggle.com/bryanpark/sudoku

[4] R.Soto, B.Crawford, C. Galleguillos, S. Misrak and E.Olguin,'Solving Sudokus via Metaheuristic and AC-3',2014, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7068127&tag=1

[5] Sankhadeep Chatterjee, Saubhik Paladhi, Raktim Chakraborty, "A Comparative Study On The Performance Characteristics Of Sudoku Solving Algorithms", International Organization of Scientific Research Journal of Computer Engineering (IOSR-JCE),e-ISSN: 2278-0661,p-ISSN: 2278-8727, Vol.16, Issue 5, Sept-Oct 2014.

[6] Our Github Link, https://github.com/kaanberke/SudokuSolver