# Comp 3106 - Project Report
# Group 28
# Member : Shawnia Noel [101207361]

## Front Matter

Title of project : Sudoku Generator (with set difficulty levels)
Course code : Comp 3106
Group 28; Member; Shawnia Noel - 101207361

## Statement of contributions

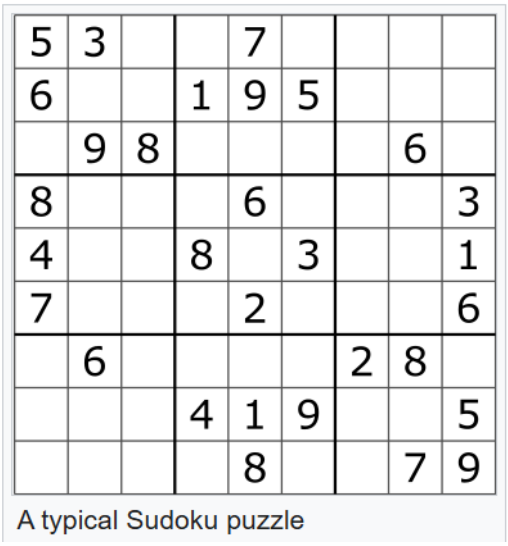
N/A - project completed alone

## Introduction

### *Background and motivation for the project*

Sudoku is a puzzle game that essentially revolves around putting numbers in a grid. It is a logic game in the sense that every input in the grid can be made from deductions. It is often said in the sudoku community that you should never guess in a sudoku or “if you are guessing, you are doing it wrong”.

A sudoku puzzle is made up of 9 rows and 9 columns in a grid. Those rows and columns are grouped together by 3, thus creating 9 subgrids. To solve a sudoku, one must fill all the 81 individual cells with a digit from 1 to 9, that is this set of digits; {1, 2, 3, 4, 5, 6, 7, 8, 9}.
However a sudoku cannot be filled with numbers randomly, the allocation of numbers must be done in such a way that it conforms to the rule of sudoku.
The rule of sudoku is as follows: A digit (which is from the set of 1 - 9) can only appear once in a row, once in a column and once in a subgrid. Another way of formulating this is that any given row, column or subgrid must have only one instance of the numbers 1 - 9, no repeat of any digit is allowed.
Below is a picture of an average Sudoku grid taken from the Wikipedia page. [1]



A typical Sudoku puzzle

As such, solving a sudoku is essentially a constraint satisfaction problem. Constraint satisfaction problem (CSP) is a subset of AI where we have to reach a terminal state through the use of a set of variables. The goal is to reach a state where all the variables have a value that does not violate any constraints/restrictions on them. In the case of sudoku all variables along the same row or along the same column or in the same subgrid have all the Alldiff constraint, meaning they should all be of different values. Moreover, in relation to agents and graph search, we will be using heuristic search to solve and generate the sudoku.

So solving a sudoku is a CSP, what about creating one ? Well, it turns out that to create a sudoku you have to essentially solve a sudoku. So generating a sudoku is also a CSP.
The project's approach to generating a sudoku is as follows;

- Take an empty grid and fill in with numbers while respecting the rules of sudoku (the constraints). In brief, this is generating a solved sudoku puzzle.
- Take the solved sudoku and remove elements from the grid.

Creating a solved sudoku is in essence solving a sudoku with just the least amount of constraints on the numbers since the grid is totally empty.
The tricky part comes in removing elements from the grid. An important aspect of a sudoku puzzle game is that it should have only one unique solution to it. If one is given a sudoku to solve and that sudoku has more than one solution then it is simply not a valid sudoku puzzle. The only way to make sure that a sudoku has 1 unique solution is to attempt to solve it with every combination of numbers allowed within the constraints. At the end only 1 sequence/combination of numbers will not violate any constraints of the 81 variables (cells).

My motivation for this project is simply my interest in Sudoku. I started solving sudokus when I was around eight, in the kids' section of the newspaper. At that time, my family did not own an internet line so the only entertainment available was tv or newspaper. I was not keen on watching sports with my dad so I settled on reading the newspaper and doing the minigames in there. As time went on, sudoku became a passion of mine and now I even have a dedicated

sudoku app on my phone. When the professor suggested doing a project based on our personal interest, I just had to decide between two options; F1 and sudoku. The F1 project would have been a race strategy project but there are too many variables involved in that and also variables which are not shared to the general public such as tyre degradation and track evolution. As such I felt that any model of this project would not be much representative of the race strategy algorithm used in real life. Hence, the decision was simple and I picked Sudoku. I considered the sudoku-solving problem way too common so I went down the route less traveled; creating a sudoku puzzle. In this era of ads, monetization and privacy issues, I figured having my own sudoku algorithm would be pretty neat since I would not have to rely on any apps or websites anymore and would not have to worry about the aforementioned issues.

## *Related prior work*

In the field of CSP, this was actually my first time tackling such a problem. Before that I did works on agents, graph search, heuristic searches as per the Part 1 of the course teachings. The chapter 6 of the course text book is on Constraint Satisfaction problems which rely on knowledge of the topics found in the past chapters.To educate myself on the topic I read that text book chapter and learned about CSP and its relevant algorithms/ techniques involved.

The next issue I ran into was how to judge the level of the difficulty of a sudoku, which is colloquially called grading a sudoku. I researched the ways on how grading is done but unfortunately there is not an agreed upon worldwide standard. From what I understand, generating sudokus is a commercial problem so apps and websites, and other commercial entities keep the way they do it guarded. Nevertheless I was able to find two broad ideas when it comes to grading which is evaluation through the techniques and the difficulty of those techniques a human solver would have to employ to solve the sudoku and the depth of recursion (iteration count) of the sudoku solving algorithm. I decided on the second option since the first one requires you to code for every technique of sudoku which I admit is beyond my skill set as there are techniques that I myself do not understand and cannot recognise in a sudoku.

To implement the second option, I was able to find a website [2] which took reference from a research paper. That research paper [3] found the link between recursion depth of a sudoku solver utilizing arc consistency algorithm with domain splitting and the difficulty of sudoku puzzles. The author of the website [2] used that idea and was able to find a strong correlation between the difficulty of a sudoku puzzle on websites like sudoku.com [4] and recursion depth of the solving algorithm. I used that same idea to find a correlation between the difficulty of sudoku puzzles on sudoku.com and the natural logarithm of the iteration count of my solving algorithm.

As for solving algorithms, I worked on a lot of search techniques on CSP. I wrote algorithms using backtracking, forward checking, arc consistency and minimum remaining value. For the final implementation I used all of those techniques except arc consistency as the arc consistency algorithm I wrote would not work as expected and was stuck in some infinite loop.

### *Statement of objectives*

The main objective of the project was to explore more topics on A.I that were not tackled in class. Also I was able to gain experience on CSP in general. CSP is a powerful technique usually used for job scheduling.
Other objectives of the project include:

- Designing an algorithm that creates a valid sudoku puzzle with a unique solution.
- Learn new search algorithms and heuristics techniques to solve state problems.
- Implement backtracking search algorithms with optimizations like forward checking and minimum remaining values heuristics.
- Find a valid correlation between the recursion depth of the solving algorithm and the sudoku puzzles found on sudoku.com [4] website.
- After establishing correlation, use appropriate values to establish thresholds for specific level of difficulties.

# Methods

### *Methods from Artificial Intelligence used*

The project uses methods generally involved in Constraint Satisfaction Problem. There is a set of variables, domains and constraints. Each cell is a variable so there are 81 variables in all. Each cell can take any number from 1 to 9, so each cell has a respective discrete finite domain with respect to the constraints. Each variable has constraints on it which is of the type AllDiff (all different) constraint.

Since the minimum clues a Sudoku can have is 17, we do not actually have to consider all the variables when solving a sudoku, at worst we consider 64 variables. We apply constraint propagation, a basic inference algorithm, when constructing the domains of variables which do not already have a set value so that when the algorithm tries each number to get a complete assignment, it rarely has to go through the whole set of nine digits.

When it comes to filling the empty grid, again the algorithm does not actually consider all the 81 variables at the same time. The grid is filled row by row with some forward checking to look at future subgrids along the row. So for assigning each variable, the variable considers about ⅓ of the 81 variables. The algorithm more or less creates these multiple batches of 33 variables and establishes local consistency. Thus, also reducing the space complexity. After looping over all these batches, at some point, it establishes global consistency and we have a valid solution. (also called consistent complete assignment). As such we can achieve a time complexity that is lesser than just normally comparing 81 variables.

For the techniques, the algorithm makes use of backtracking search with inference algorithms, both of which are common algorithms used in CSP. The main idea here is that backtracking causes multiple frames to be piled on the stack and although the risk of stack overflow is very minimal, in general we want to add the least amount of frames onto the stack as popping and pushing frames on the stack have an overhead.
For filling the grid, we use a backtracking search together with the inference algorithm of forward checking. We first apply a basic inference algorithm called constraint propagation to get the minimal domain of a variable. Then we utilise a heuristic to prioritise numbers that appear in future subgrids so that they appear earlier in the row and as such reduce the need for backtracking. When a conflict with constraints is found, the algorithm backtrack and tries other numbers for the previous cell we filled. If it cannot find one, it backtracks again and performs the same actions. Eventually it will reach an appropriate solution that does not violate any constraint on any variable. Doing constraint propagation and forward checking help alleviate the time complexity of the algorithm as we do not waste time looping over numbers that are not candidates and thus greatly reduce the need for backtracking which is of itself computationally expensive.

For solving the grid, the algorithm still uses the backtracking algorithm but this time we use a minimum remaining value (MRV) heuristic which is also a technique of CSP. We first use a basic constraint propagation inference to construct the minimal domain for each unassigned variable, then we use the MRV heuristic to first consider the variables which have the smallest domain, that is, those which have the least possible option of numbers. Doing MRV has somewhat the same effect as arc consistency in the sense that tackling variables that have the smallest domain helps reduce the domain of other variables. This process of narrowing down the possible values for variables helps minimize the occurrence of backtracking which has a high computational cost.

The solving algorithm is also slightly modified to push the solver to try to find other solutions beyond the first one. It backtracks and attempts other numbers and if a second solution is found it returns immediately. This is the method we use to ensure that the sudoku still has a unique value. At the end of the solve, the solving algorithm returns a solution count of either 1 or 2. The code implementation is only given the green light to clear out the chosen cell when the solution count is returned as 1. Moreover, we keep track of how many times the solver recursively calls itself and use that number to determine what level of difficulty is the current generated sudoku puzzle.

## *Dataset*

There is not a specific dataset used. I simply loaded the website sudoku.com and picked a sample of random sudoku puzzles for each difficulty level. After that, I manually input the puzzle arrangements for multiple sudokus one by one into my algorithm. Then I ran my solver and recorded the iteration count it took to find a solution. I also took an arrangement of invalid

sudoku puzzles (non-unique or no  solution) and ran it through the solver to make sure it could accurately detect when a sudoku had more than 1 solution or no solution at all. I did manual input instead of using any external python library or sudoku API because I did not want to introduce any unnecessary dependencies to the project.

## *Validation strategy or experiments*

From my research on the topic of generating sudoku, I stumbled upon a website which itself took inspiration from a research paper [Refer to 2 and 3 in references section] that found a positive and strong correlation between the recursion depth of the solving algorithm and the difficulty level of sudoku puzzles on sudoku.com and websudoku.com which are the most well known and most used online website to play sudoku by the average person.Taking inspiration from that, I decided to use the method to generate sudoku of a specific difficulty.

To assess my correlation I only used puzzles from the sudoku.com [4] websites. The reason for this is because this site is just the most popular one and is a frame of reference of sudoku difficulty for the average sudoku enjoyer. I took a sample of 5 puzzles from that site for difficulty levels easy, medium, hard and expert. I manually input the configuration of those puzzles to the solver algorithm and ran it. I recorded the iteration count for each of the puzzle solves and stored it in a csv file. To that csv file I also added a column for the natural logarithm of the iteration count. I then assessed the degree of correlation between the iteration count and the difficulty level using an online website [5] and got a correlation value of 0.6512 and a p value high enough for the correlation relationship to be deemed relevant at the 1 %. As such with these results I was able to confirm that there was indeed a correlation between the iteration count of my solver algorithm and the difficulty level of the sudokus on the website sudoku.com. From there I used a median value of the natural logarithm of the iteration count in each level from the report to establish thresholds of iteration for each difficulty level for the sudoku generator algorithm. Essentially, given a difficulty level, the algorithm first creates a solved sudoku and then keeps removing elements from that sudoku until the sudoku puzzle left is solved within a certain threshold value of the natural logarithm of iteration counts of the solving algorithm.

Next I fed the solving algorithm invalid sudokus with more than one solution and was able to confirm that the algorithm is able to detect when a sudoku is invalid with an accuracy of 100 %.

The final part was testing that the output of the sudoku generator are actually solvable sudokus with one unique solution at all the difficulty levels. To test that I took ten samples (ten sudokus output) at the four difficulty levels (that is 40 sudokus in all) and manually input the data one by one for each sudoku in on an online sudoku solver [6] (I used this website: https://www.sudokuwiki.org/sudoku.htm). This specific solver has a ‘solution count’ option which I used to check the number of solutions to the sudokus generated. I was able to record that the sudoku generator algorithm generates valid sudoku puzzles with only one unique solution with an accuracy of 100 %.

# Results

## *Quantitative Results*

I calculated the correlation coefficient between the natural logarithm (ln) of the iteration count (recursion depth) and the sudoku puzzles' difficulty on the sudoku.com website. This came up to 0.6512 indicating a positive moderate relation. The original website [2] from which I drew this idea had the same correlation at a much higher coefficient of 0.9, though they used a much bigger sample size of at least a hundred puzzles for each level. Furthermore, I calculated the p-value for the hypothesis of the existence of the relation between the two which came up to be 0.001386. Through this p value I was able to prove that the hypothesis is highly significant (significant at the 1%) and as such with the support of both the r-value and p-value could confirm that the difficulty of the sudoku puzzles on the sudoku.com website do in fact rely on the recursion depth of the solver, like previously observed by the online website's author. [2]
From the report of the natural logarithm (of the iteration count) values in the correlation csv file I created, I used a median value of the values in each level to get the following thresholds;

- Easy level; natural logarithm of iteration count should be at least 3.7
- Medium level; natural logarithm of iteration count should be at least 3.83
- Hard level; natural logarithm of iteration count should be at least 4.44
- Expert level; natural logarithm of iteration count should be at least 5.70

Through validation experiments, I was able to evaluate that the accuracy of the algorithm in detecting invalid sudoku puzzles (non-unique solution) is 100% . The accuracy of the algorithm in generating valid sudoku puzzles (unique/only one solution) is also 100%.
Note that the algorithm is more focuses on detecting whether multiple solutions exist rather than whether a solution even exists (that is possibility of sudoku having no solution) because we start from a solved sudoku in the first place so whichever sudoku we create from that is guaranteed to have at least one solution.

## *Qualitative Results*

The sudoku generator algorithm reliably creates valid sudoku puzzles, each with a unique solution and of difficulty levels that are consistent to those found on sudoku.com, a popular website used by the average casual sudoku player. It is able to do so because we have observed a relationship between the recursion depth of the solver algorithm and the difficulty level of sudoku produced in sudoku.com. The algorithm does this by utilizing backtracking which is admittedly computationally expensive but uses inference algorithms, like constraint propagation and forward checking, and heuristics like minimum-remaining values heuristics to minimize the need for backtracking thereby reducing the time complexity and computational cost.The algorithm has proved to be accurate in producing valid sudokus of difficulty level specified by the user. The sudoku generator algorithm as a whole works very quickly with no stuttering and sudoku puzzles are created instantaneously. The algorithm was run multiple times back to back and there was no lag observed over time.

# Discussion

## *Limitations of the work and directions for future work*

The sample size used for correlation analysis and validation is quite small by industry standards. More samples should be used for each category of puzzle, at least 100 samples would be an appropriate size. An efficient automated way should also be used for validation and evaluation beyond manual input to an online solver.

The algorithm so far works for standard 9x9 sudoku puzzles though it can be extrapolated to work on 4x4 and 16x16 or other $nxn$ sudoku puzzles. However, the algorithm only works for the classical way of playing sudoku and not on sudoku variants like killer sudoku or samurai sudoku.

The sudoku puzzles are graded to a difficulty level based on the recursion depth which although do have a moderate to strong correlation is not always 100 % the case. There can be very difficult puzzles that can have relatively low iteration count which will then cause them to be inaccurately graded to a lower level, vice-versa can also happen. Admittedly a better way to grade sudoku would be to find out what deduction/sudoku techniques a human solver would have to utilize in order to solve it. However due to time constraints and high complexity of certain techniques like X-Y wing and swordfish, I was unable to code for such a grading algorithm.

For future work, a direction that definitely should be explored would be that grading algorithm The new grading algorithm could rate the puzzle according to the sudoku techniques used by a human solver and their associated difficulty. The algorithm would have to accurately detect when the scenario for the techniques occur, especially for the very advanced and niche techniques. Perhaps, furthermore a neural network could be trained to recognize such scenarios and could then be used as a grader.
Algorithms for several sudoku variants can also be worked upon in future work. I think killer sudoku or samurai sudoku would be a pretty interesting project to work upon.

## *Implications of the work*

This project has allowed me to gain hands-on knowledge on another topic of AI; constraint satisfaction problem. CSP is a powerful technique often used for task scheduling and as such I believe it would be helpful for another interest of mine which is supply chain management.
I'm confident that this knowledge in CSP will be a good foundation for projects related to supply chain management and integrated supply chain.

All in all, the scope and objectives of the project were achieved. The algorithm is able to always create sudoku for a specific level set by the user. The difficulty levels that the sudoku puzzles are created are consistent to those found on the site sudoku.com[4]. Consequently, regular

users of the site will find the generator algorithm's puzzle difficulty levels are consistent with those found on the site. The project's algorithm also always creates sudokus with one unique solution that does not violate any constraints imposed by the Sudoku game rules.

It accomplishes this by using the backtracking search method which is known to be computationally expensive but owing to  inference algorithms and heuristics used, we are able to greatly reduce the space and time complexity that a generic backtracking search would have taken.

# References


1. Stellmach, T. (2017, April 8). Sudoku image. Retrieved from: https://en.wikipedia.org/wiki/Sudoku . Picture is also originally hosted on: https://en.wikipedia.org/wiki/Sudoku#/media/File:Sudoku_Puzzle_by_L2G-20050714_standardized_layout.svg

2. Nick, T. (2024, June 25). *Generating sudokus for fun and no profit*. Retrieved from https://tn1ck.com/blog/how-to-generate-sudokus

3. Fatemi, B., Kazemi, S. M., & Mehrasa, N. (2014). *Rating and generating sudoku puzzles based on constraint satisfaction problems*. World Academy of Science, Engineering and Technology International Journal of Computer, Information, Systems and Control Engineering Vol:8 No:10. Retrieved from https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fc78d54ae5d5234c9fb229d6a2cd2ef5af181e70

4. Sudoku.com. (n.d). Retrieved from:  https://sudoku.com/

5. Social Science Statistics. (n.d.). *Pearson correlation coefficient calculator*. Retrieved from: https://www.socscistatistics.com/tests/pearson/default2.aspx

6. Stuart, A. (n.d). Sudoku Solver. Retrieved from: https://www.sudokuwiki.org/sudoku.htm