Table of Contents

1. Introduction

Sudoku, a puzzle known as combinational and logic which involves placement of numbers since the 20th century, is a problem that is widespread. The main goal of the puzzle is very simple: we should figure out where to put the digits in '9×9' grid in a way that every one of the columns, rows and additionally the 3×3 blocks that describes the grid (actually boxes, blocks, regions) has digits ranging from '1' up to '9'. The crossword puzzle presents a set of intricate and solvable problems; as a result, the people are encouraged to use their critical thinking and problem solving skills.

The coursework project report is on SUDOKU Design which explains the different ways in which we will solve this problem using Python. Instead of focusing only on the solving of a particular Sudoku and ignoring different solution states, which can be either there are no solutions, there is one or multiple unique solutions for the same puzzle. However, the generator is the mechanism that serves to provide new and unpredictable Sudoku puzzles with only a one time solution, which is always a new challenge for the users. The implementation of the combination of the foundations of applicable algorithmic ideas with modern software development formats results in a product that is still informative although remains interactive.
For fig. (https://www.rd.com/list/printable-sudoku-puzzles/)

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

1.1 History of Sudoku

When French puzzle setters started experimenting with deleting numbers from magic squares in the late 19th century, number problems started to emerge in newspapers. On November 19, 1892, the Paris daily "Le Siècle" published a partially completed 9x9 magic square featuring 3x3 subsquares. It wasn't a Sudoku because it had double-digit numbers and needed arithmetic to be solved instead of reasoning, but it did have one important thing in common: every row, column, and subsquare tallied up to the same amount. (http://www.diva-portal.org/smash/get/diva2:811020/FULLTEXT01.pdf)

Rival of Le Siècle, La France, improved the puzzle on July 6, 1895, making it nearly identical to a contemporary Sudoku and dubbed it "carré magique diabolique" (or "diabolical magic square"). The 9×9 magic square problem was simplified to include simply the numbers 1-9 in each row, column, and broken diagonal, but no subsquares were marked. The unmarked 3×3 subsquares contained the numbers 1-9, and the broken diagonals requirement resulted in a single solution.

These weekly puzzles were a staple of French publications like L'Écho de Paris for approximately a decade before disappearing around the time of World War I.

1.2 Modernization

The earliest known examples of contemporary Sudoku were Number Place, published by Dell Magazines in 1979. Howard Garns, a 74-year-old retired architect and freelance puzzle constructor from Connersville, Indiana, most likely invented the modern Sudoku anonymously.[1] Garns's name was consistently listed as a contributor in Dell Pencil Puzzles and Word Games issues that had Number Place, and it was consistently missing from those that did not.[10] He passed away in 1989 before witnessing his brainchild become an international sensation.[10] It is unknown whether Garns was familiar with any of the French newspapers mentioned above.

In April 1984, Maki Kaji, the president of Nikoli puzzle company, introduced the puzzle to the Japanese public through the paper Monthly Nikolist. The puzzle's translation, Sūji wa dokushin ni kagiru, means "the digits must be single" or "the digits are limited to one occurrence" (dokushin in Japanese means "unmarried person"). Later on, the name was shortened to Sudoku (数独), which is a simplified form of the name that only uses the first kanji of compound phrases.[10] Although the term "Sudoku" is a registered trademark in Japan[11], the puzzle is more commonly known as Number Place (Nanbāpurēsu) or, colloquially, Num(ber) Pla(ce) (ナンプレ, Nanpure). Two changes were made to the puzzles by Nikoli in 1986: first, the number of givens was limited to 32, and second, the puzzles were made "symmetrical" by distributing the givens among cells that were symmetrical in rotation. These days, respectable Japanese publications like the Asahi Shimbun publish it.

2. Overview of the available solutions

The computational way that solves Sudoku puzzles evolves through different algorithms and methodologies with each having its own unique advantages and challenges. Below is an overview of several prominent solutions and alternatives explored in computational and algorithmic literature: Below is an overview of several prominent solutions and also alternatives explored in the computational and algorithmic literature:

Backtracking Algorithm

The backtracking algorithm is a brute-force recursive tactic that solves the Sudoku puzzles. It fills the gaps in the empty cells one by one until the current board status remains surplus. If the number arrangement results in a contradiction at a later stage, it goes back to the previous step, then exchanges the number being considered with its next possible option. Although it is straightforward, the backtracking algorithm is an extremely efficient tool for solving the classical Sudoku puzzles but can be inefficient if designed to override a brute force approach. (https://www.geeksforgeeks.org/backtracking-algorithms/)

Constraint Propagation

Constraint propagation algorithms come a step further with using the rules of Sudoku to get rid of any numbers that might be a possibility of each cell, greatly narrowing down the search space. Naked pairs and « hidden singles » are among the methods applied for similar purposes. The first one allows you to get rid of the cells with a single possible number. The second one helps to become informed about the candidates' lists of candidate cells. Constraint propagation is very helpful to solve many puzzles but it still needs to be used together with many other methods when encountering the most difficult ones. (https://www.ibm.com/docs/en/icos/20.1.0?topic=optimizer-constraint-propagation#:~:text=over%20this%20variable.-,Constraint%20propagation%20is%20the%20process%20of%20communicating%20the%20domain%20reduction,communicated%20to%20the%20appropriate%20constraints.)

Stochastic Algorithms

Stochastic algorithms, including Genetic Algorithms and Simulated Annealing, exploit many random techniques for the solving of Sudoku puzzles. These means are particularly helpful for working out the puzzles' solution space in a non-linear fashion, which may result in faster solution finding operations for really complicated puzzles. Nevertheless, these algorithms are prone to high complexity and also cost computations of implementation. (https://www.sciencedirect.com/topics/computer-science/stochastic-algorithm#:~:text=Stochastic%20algorithms%20are%20particularly%20interesting,to%20solve%20various%20mathematical%20problems)

Knuth's Algorithm (the Exact Cover Problem and the Algorithm X)

Not only Donald Knuth's Algorithm X, but it can also be the exact cover problem which may be applied to Sudoku. The algorithm uses the exact cover problem formulation of the Sudoku constraints involving the row, column, box, and also cell uniqueness as well as the "Dancing Links" technique that is efficient

for backtracking and ultimately solves the problem. This approach is very clean and effective, but it can realize any Sudoku puzzle solution. However, the complexity of its implementation is much higher than that of the more simple algorithms. (https://rosettacode.org/wiki/Knuth%27s_algorithm_S)

Machine Learning Approaches

New approaches to the solution of Sudoku with the aid of machine learning technologies have emerged lately. In the training of neural networks, it can be foretold about the numbers of a Sudoku puzzle with the help of the patterns from a big quantity of solved puzzles. The logical problem solving approaches discussed below are not of the traditional algorithmic type but possess wonderful elements in showing us the way AI does the logical problem solving task.

2.1 Comparative Analysis

Every of them implies a different level of efficiency, implementation complexity, and also solving abilities. The backtracking algorithm excels exactly for its simplicity and wide utility, thus it is well suited to serve as a very basic solving. Constraint propagation and stochastic methods offer more intricate ways of solving the harder problems while the Algorithm X and the machine learning methods are the modern technologies that are used in solving the Sudoku.

3. Overview of the available researches:

Berggren, Patrik, and David Nilsson. "A study of Sudoku solving algorithms." Royal Institute of Technology, Stockholm (2012).
(https://www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2012/rapport/berggren_patrik_OCH_nilsson_david_K12011.pdf)

The study features analysis of three algorithms that can be used to solve Sudoku. Besides solving-difficulty rating, co-builder-ability, and eligibility for concurrency, the study focuses on solution ability. This consideration is made at the level of each algorithm and also in comparison with other algorithms. Three types of algorithms are evaluated: Boltzmann machines, rule-based, and backtracking. The Boltzmann machine works on the easier clones of the 17-clue puzzles in the database and assessments are made by timing how long it takes to recover the needed clues for each one. Each approach is represented by solving time distributions, whereas relationships linking the algorithms are shown too. Concluding that the rule-based approach is the most efficient is the result of the research. The results show that there is a concurrent rating in the level of difficulty for both the backtracking and rule-based methods. In parallelization apply to all algorithms to different degrees, but specific implementations for heuristic approaches.
Generation has been figured out through deterministic algorithms like backtrack and rule-based.

Maji, Arnab Kumar, and Rajat Kumar Pal. "Sudoku solver using minigrid based backtracking." 2014 IEEE International Advance Computing Conference (IACC). IEEE, 2014.
(https://ieeexplore.ieee.org/abstract/document/6779291/)

Each of the classic approaches requires us to walk over 81 cells independently and execute backtracking for each one. In this study, we aim to design a minigrid-based algorithm, which means that we must walk through nine minigrids (rather than 81 cells) and execute backtracking just on them, which takes less time. Furthermore, no guessing is required, and no unnecessary computations are performed during the process.
Their proposed algorithm takes into account each of the minigrids numbered 1–9. Each minigrid may or may not provide clues in the form of numbers. We start with the first minigrid and then use the hints in the minigrids to determine which permutations of the missing integers belong to the same row and column. Then it moves on to the next minigrid and continues to analyze the remaining minigrids in a certain sequence of minigrids based on adjacency (SM). There are various conceivable SM sequences that begin with a corner minigrid (number), such as 1, and initially follow a row-major order, or the reverse.

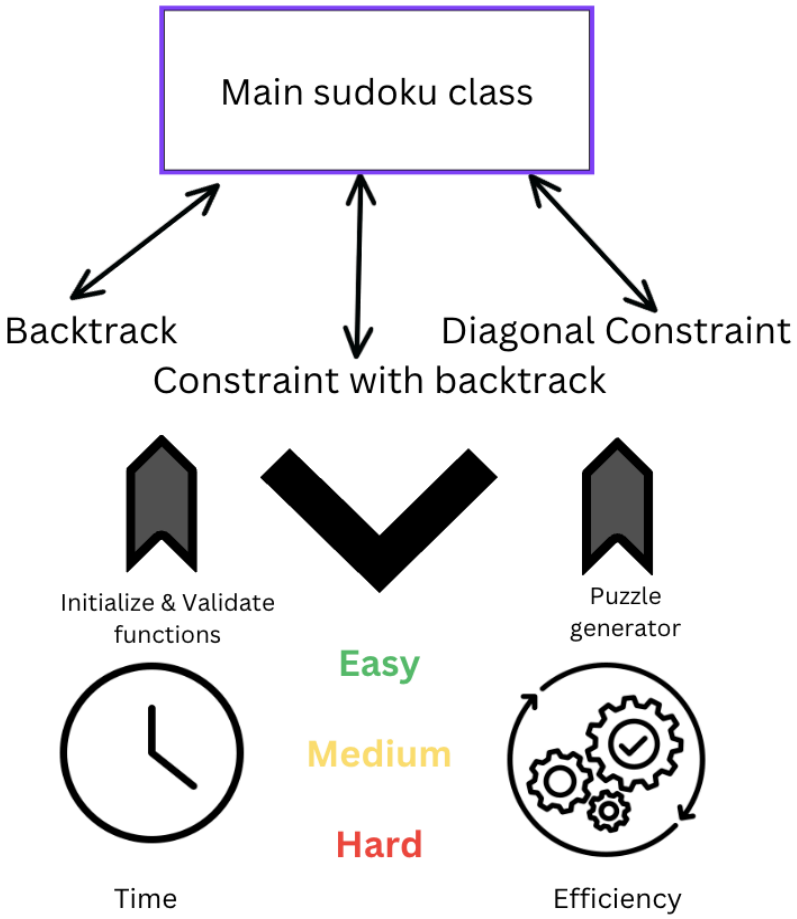
4. Design Overview

The Sudoku solver project is aimed at coming up with a broad solution to solving puzzles, that is, the regular Sudoku problems, the more difficult puzzles by applying constraint propagation, and the other kind of these puzzles, Diagonal Sudoku, that have diagonal constraints.
We have a hybrid solution, getting the advantages of using the backtracking and the constraint propagation methodologies together. In a contradiction situation, the backtracking algorithm intends to place subsequent valid digits in the cells as well as un-doing any previous actions. Application of Sudoku rules to eliminate unlikely numbers by means of constraint propagation technique serves to narrow down the problem area, whereas AI-based algorithms could theoretically outperform it (constraint propagation) by means of solutions provided with given clues. This combined effect also adds reliability and convenience when completing Sudoku puzzles.

Also we recommend adding a parameter to the solver class that users can set any one of them to get a solution to the Sudoku puzzle the conventional way or Diagonal Sudoku way. In a Diagonal Sudoku, only nine numbers from 1 to 9 are available, and the two diagonals, the main diagonal being read from left to right and the secondary one from right to left, are each made up of four. The implementation of this is done by creating the solver with an added constraint to it.

5. Architecture

The problem-solving, question-checking, and puzzle-making logic core encapsulated in the class "Sudoku" is what determines the overall structure of the project. This course is versatile, so many types of solving techniques are applicable depending on the type of difficulty faced, or the level of efficiency needed.



5.1 The architecture has following components:

a. Core Components:

The system made by parts is called the SudokuSolver, the whole system is.
Sudoku Class: The fundamental class of a given Sudoku problem is its Sudoku class. The board representation can determine if the expression is of standard form along with the printed value of the function & result as well as deciding whether it is a complete expression.

b. Algorithms:

Backtracking Algorithm Module: This module tries to fill the board with numbers through use of recursion, however, it backs-track when it comes across an invalid condition.
Inputs: The condition of a present Sudoku.
Results: Solve the puzzle and get a Sudoku board with all completed squares or just a message that puzzle can not be solved.

A pre-processing part known as the Constraint Propagation Module uses the Sudoku rules to eliminate values that would cause an impossible number to appear in each cell, thus narrowing the search field.
Input: What the Sudoku board looks like right now.
Result: The options available for every tile cell are decreased as a result.

Sudoku in diagonals: Additional backtracking enhancement is provided by the diagonal sudoku module. For the diagonal requirements that are specific to the Diagonal Sudoku puzzles, more significant tests should be added.
Input: The diagonal Sudoku grid is incredibly readable.
Result: A "The riddle cannot be solved" message or a Diamond Sudoku puzzle would be suitable choices for the emblem.

c. Supportive Functions:

Initialization and Validation functions: The following two functions generate new challenges, as well as confirm the already existing situation on the board.
Puzzle Generator: A component that can produce Sudoku problems that can be solved by starting with a board that has no numbers and then selected band elimination until the only possible solution is left.

6. Implementation Analysis

6.1 Standard Problem

The class `SudokuSolver` is named `SudokuSolver` from the method and has been realized. The updated ‘SudokuSolver’ class in our improved methodology employs a hybrid approach that involves both the backtracking algorithm and the constraint propagation technique to tackle Sudoku problems faster. The class constructor is, however, rather flexible and can accept inputs of the grid size (9x9 being the default for a typical sudoku), an optional pre-populated board, and the mode which picks the problem as being 'standard' or 'Diagonal Sudoku', catering for the additional difficulty of the latter.

‘isValid’ method is used to check whether a number satisfies Sudoku rules for having a number within a particular cell or not. This guarantees that the same number is not used in any row, column, or super-position element. The horizontal Sudoku also computes diagonal lines, the primary one as well as the secondary one.

The "solution()" function decides between the horizontal and the oblique operations of Sudoku. This approach is based on the backtracking technique which uses repeated recursion in an attempt to fit a solution in the blanks until the solution is found. The validation degree has a maximum in contrary mode, and there is a uniqueness on the diagonals of the table.

The generateSudokuPuzzle function takes an optional number of clues that create a new blank Sudoku puzzle by first filling out a completed puzzle and then re-eliminating pieces to leave the remaining puzzle with only unique solutions. (https://github.com/Mercrist/Sudoku-GUI)

6.2 Constraint Propagation

Constraint propagation integration enhances the system's effectiveness of solving. The method **applyConstraintPropagation**, which successively applies Sudoku rules, is first called before the backtracking process begins. This method eliminates all impossible numbers from the potential cell number. This phase of preprocessing leads to a reduced search space for the backtracking algorithm, helping it focus on the solution more and consequently reduce the search time. It's indispensable to remain the **validPlacement** method for verifying whether a number is valid to enter in a given cell and preventing duplication inside a row, column or subgrid. The guaranteed consequence of this verification is the integrity of the main and secondary diagonals for the Diagonal Sudoku variant, which keeps it in line with its unique constraints.
The mode used decides the algorithm for **'solve'** among the vertical mode and diagonal Sudoku solving methods. The new method employs the constraint propagation method to provide a preamble for backtracking, pinning more faith on a more advanced algorithm for filling empty cells with valid digits. Backtracking is a process of a recursive type that entails a step by step logical searching for the potential answers (possibly also for incorrect entries) where inconsistencies are revealed and backtracking takes place. In the diagonal Sudoku mode, the solver guarantees that there are unique solutions and follow its variant special rules along the diagonals.

By taking advantage of the advances made with constraint propagation, the **generateSudokuPuzzle** function which now can take the number of clues as the input parameter and produce a new Sudoku puzzle with a specified number of clues in addition to being faster has been improved. It is so that after puzzle completion, the only manner of solving the final puzzle is the process of very cautious elimination of numbers after the completed the fully populated table. Constraint propagation is injected to the puzzle generating phase in order to enhance this procedure even further. With this you can be sure that the puzzle is as solvable as possible at the beginning of the process so that the solution of the whole puzzle does not depend on the clues removal.

6.3 Implementation Analysis with code explanation

We have few important functions to look at in each class:

For Sudoku with backtracking class:

‘validPlacement’

Boundary check if the number is on the right side of a legal grid by row and col.
It guarantees that the other cell, column, or 3x3 subgrid doesn't have the given value.
Returns False, meaning the filling violates the Sudoku rules, and True if the placement is valid.

‘searchEmptyLocation’

For the first empty cell, marked 0, searches on the board and returns its location (row, col).
If no empty cells are found, it means that the board is dedicated and, thus, returns None.

‘Solve’

This is the essence of the solver, going through a backtracking course of action.
Then it checks out if the cell is empty. if someone exists the puzzle is solved, and it returns True.
It makes all possible attempts by trying to fit every possible number (1-9) and recursively calls the solve function to try and go through the remaining parts of the board.
If a placement results in no solution, go back to resetting the cell (backtracking) and tries the successive ones.
Returns False if the current empty cell is unable to receive any valid placement, indicating the need for more backtracking.

‘generateSudokuPuzzle’

It starts by using the solve method to make a fully solved board.
It continues by randomly erasing cell numbers, and always maintaining the number of puzzle solutions (initially filled cells) to the specified parameter.
It carefully get rid of the cell as we descend that is matching the clue parameters. The objective is to find the right balance between puzzle solvable & challenging.

For Diagonal Sudoku class:

‘isValid’: The isValid method ensures that the number (num) is correct to be placed in the given cell on the Sudoku board, without any violation of the rules of the game. Then it checks whether num is already present in the row or column. Then, the function checks whether num is in one of the same 3x3 subgrid. For strong mode (x-sudoku) additionally num is checked as an element of one of two main diagonals and uniqueness of the element is required here as well. If num satisfies all of the mentioned checks (not available in row, column, subgrid, or diagonal as appropriate), we assume that placing num is correct (True), otherwise it's incorrect (False).

‘solve’: Jumps into a recursive backtracking algorithm. It sets the focus on cell locations in which there are zeros (empty spots) then proceeds to put the valid numbers in them (1-9). The AI algorithm will delete a cell if a particular number fails to solve a state, and will then move to the next number. This cycle continues until the grid is either solved or considered unsolvable by the solver.

‘validateSolution’: the idea of this method stub is to offer a spot for putting validation into effect of solutions, so that they follow the rules of Sudoku. So it is the same as the previous class of standard sudoku.

For Constraint Propagation class:

'searchEmptyLocation' function loops through every row and column of the Sudoku board searching for the first empty cell marked by 0. If the mutation causes it to find a cell like that, it will immediately return the coordinates in the form of (row, column). This is one of the major advantages why we use a backtracking algorithm to determine which numbers should be tried next. When making a full board (there are no empty cells), the function is finished and it will not return any coordinates, it means the puzzle is solved or it needs further processing.

The **'applyConstraintPropagation'** function loops through all empty cells in the Sudoku puzzle to remove the list of possible values step by step. It threads via a cycle that would end up once nothing can be simplified more (when changed becomes False). For every empty cell it computes the possible solutions without breaking the rules of the Sudoku using the possibleValues procedure. In case the unique value is encountered, the cell is filled with the value and is marked changed to True to indicate the progress, thus, to generate another iteration. Thus, this preprocessing is to simplify the problem until other effective methods like backtracking can be used.

'possibleValues' function defines all the (possible) numbers that are allowed to be put in any Sudoku board cell taking into account the current state of the puzzle. The number range from 1 to size is passed around in a group of numbers that may accommodate a certain cell. After that it removes numbers that exist in the same row and column. In addition, the numbers added in the same 3x3 subgrid are deleted in order to obey the sudoku rules. The cells remaining in the grid correspond to the valid positions that can be occupied by the displayed value at [row, col].

Using constraint propagation to simplify the puzzle, the **"solve"** approach starts the puzzle-solving process at the first run. After then, the function **"solveRecursive,"** which employs the backtracking algorithm, is concluded. By searching for an empty cell, this algorithm attempts to enter a number between 1 and the size.Next, the number fills the next vacant cell if Eq. is satisfied (as indicated by validPlacement). Until a figure fits or points in the right direction, it won't go further. The device resets, or returns the cell to 0, then attempts a different number if nothing else appears to work. The riddle continues until it is either solved or deemed impossible to solve.(https://github.com/Gecode/gecode)

7. Testing with Examples

7.1 Methodology

Testing was conducted in a systematic way, starting with simple problems in order to tell whether the solver is able to solve those problems and then continuing with harder and more complex Diagonal Sudoku puzzles in order to test its enhanced capacities. Gazing at every puzzle, the solver was determined that there should be a solution found if possible(valid solution where required) and the time taken for it should be reasonable enough.

7.2 Test cases:

Easy Puzzle: A normal mixed number hard puzzle with evident solution, which should just check the backtracking solver algorithm functionality.

Medium Puzzle: A puzzle challenging the efficiency of the backtracking solver when executed without the help of a solver that simplifies the problem with the help of the propagation of constraints.

Hard Puzzle: A sophisticated problem that is solved by approaches that need much thinking and evaluates the performance of constraining for the time of the solve in reduction.

Diagonal Sudoku Puzzle: This jigsaw problem that has the Diagonal Constraints and tests the resolver's ability to observe the extra rules for Diagonal Sudoku problem.

7.3 Test Cases Example

a. Easy puzzle:

Puzzle: A crossword puzzle, an easy-to-follow one that is not thoroughly filled.
Result: The solvers (backtracking-only, and backtracking with constraint propagation) found the solutions quickly, demonstrating the fundamental functionality.

Easy puzzle time with standard backtrack

```
Test the Easy, Medium and Hard Sudoku pizzles

Easy Sudoku solved:
2 6 4 8 5 9 3 1 7
9 8 1 7 3 4 6 5 2
7 5 3 6 2 1 8 4 9
1 3 5 2 9 7 4 8 6
8 9 2 5 4 6 7 3 1
4 7 6 3 1 8 9 2 5
3 1 8 9 7 5 2 6 4
6 4 9 1 8 2 5 7 3
5 2 7 4 6 3 1 9 8
The time taken to Solve a Easy Sudoku :  0.0012938976287841797  seconds
```

Easy puzzle time with constraint propagation along with backtracking

```
Easy CSP Sudoku solved:
2 6 4 8 5 9 3 1 7
9 8 1 7 3 4 6 5 2
7 5 3 6 2 1 8 4 9
1 3 5 2 9 7 4 8 6
8 9 2 5 4 6 7 3 1
4 7 6 3 1 8 9 2 5
3 1 8 9 7 5 2 6 4
6 4 9 1 8 2 5 7 3
5 2 7 4 6 3 1 9 8
The time taken to Solve a Easy Sudoku :  3.504753112792969e-05  seconds
```

b. Medium Puzzle:

Puzzle: An acrylic grid with the correct number of empty fields and multiple ways to solve.
The solver with constraint propagation was faster than the basic backtracking solver. Moreover, down the line, we saw the benefits of pre-processing the board.

```
Easy CSP Sudoku solved:
2 6 4 8 5 9 3 1 7
9 8 1 7 3 4 6 5 2
7 5 3 6 2 1 8 4 9
1 3 5 2 9 7 4 8 6
8 9 2 5 4 6 7 3 1
4 7 6 3 1 8 9 2 5
3 1 8 9 7 5 2 6 4
6 4 9 1 8 2 5 7 3
5 2 7 4 6 3 1 9 8
The time taken to Solve a Easy Sudoku :  3.600120544433594e-05  seconds

Medium CSP Sudoku solved:
1 5 6 2 3 9 7 8 4
7 4 3 8 5 6 9 2 1
9 8 2 1 4 7 5 6 3
3 7 5 4 1 2 8 9 6
6 1 4 9 7 8 2 3 5
2 9 8 3 6 5 4 1 7
4 6 9 7 8 3 1 5 2
5 2 1 6 9 4 3 7 8
8 3 7 5 2 1 6 4 9
The time taken to Solve a Medium Sudoku :  3.0994415283203125e-05  seconds
```

c. Hard Puzzle:

A puzzle is a grid of possible solutions and few clues; it branches extensively and must be traced several times to reach a completed path.
The Solver implementation with constraint propagation reduced the search space considerably and, thus, it is faster than backtracking by itself!

Hard puzzle time with standard backtrack

```
Hard Sudoku solved:
1 8 7 3 6 4 2 9 5
2 3 9 5 7 8 6 4 1
5 6 4 1 2 9 3 8 7
7 1 8 9 3 2 4 5 6
4 9 6 7 1 5 8 3 2
3 2 5 8 4 6 1 7 9
8 7 1 6 9 3 5 2 4
6 5 2 4 8 7 9 1 3
9 4 3 2 5 1 7 6 8
The time taken to Solve a Hard Sudoku :  0.022342205047607422  seconds
```

Hard puzzle time with constraint propagation along with backtracking

```
Hard CSP Sudoku solved:
1 8 7 3 6 4 2 9 5
2 3 9 5 7 8 6 4 1
5 6 4 1 2 9 3 8 7
7 1 8 9 3 2 4 5 6
4 9 6 7 1 5 8 3 2
3 2 5 8 4 6 1 7 9
8 7 1 6 9 3 5 2 4
6 5 2 4 8 7 9 1 3
9 4 3 2 5 1 7 6 8
The time taken to Solve a Hard Sudoku :  3.0994415283203125e-05  seconds
```

d. Diagonal Sudoku puzzle:

Puzzle: A Sudoku puzzle with diagonals implemented, standard type.
Result: The eventually (solution) of the puzzle was sophisticated by the new diagonal rules, and the puzzle was finished elegantly and the adaptability was exhibited at the end of the solution.
(X-sudoku is diagonal puzzle)

```
X-Sudoku solved:
3 2 4 1 5 6 9 7 8
6 5 8 9 7 3 4 2 1
7 9 1 8 4 2 3 5 6
2 7 5 4 3 1 8 6 9
4 1 9 7 6 8 5 3 2
8 3 6 5 2 9 7 1 4
1 6 7 3 8 4 2 9 5
5 4 2 6 9 7 1 8 3
9 8 3 2 1 5 6 4 7
The time taken to Solve this Sudoku :  0.2500479221343994  seconds
```

8. Analysis of the Results

The process of testing reflected the merits and various applications of every problem-solving strategy applied in the course of the project.

Backtracking Alone: Well-done at these easier puzzles, while efficiency got worse when problems complexity increased. The source of its power lies in its simplicity and in the fact that, by applying it, one has a more profound and rational approach. It is evident from the screenshot that is posted in the test case examples.

Backtracking with Constraint Propagation: Ultra-high performance is presented for medium and complex assignments. With the search space being tilted ahead of the proceedings, it enabled the solver to bypass unnecessary calculations, and the results became faster in the process. This way offers the most suitable combination of this term and performance, which is why most Sudoku puzzles are based on it.

Diagonal Sudoku Feature: A solver performance does not decline in the course of addition of diagonal constraints into the system. Thus, it served as a movable finite component that caused no reduction in the solver's efficiency while allowing it to complete a wider range of puzzles.

9. Final Reflection on Evaluation & Future Works

9.1 Design Choice

One of the critical design decisions that we made was the choice to adopt a hybrid technique which merges the technique of backtracking and bottom-up processing approach (constraint propagation algorithms). This synergy is a good balance between simplicity and efficiency, allowing a Sudoku solver to be easily extensible on a wide range of Sudoku puzzles. Additionally, by including multi-mode Diagonal Sudoku, the solver grabs these puzzles and hence, the seekers that want to solve more complicated riddles.

Despite that, the prompt flexibility in algorithm choosing is something that can be explored in the future. Whereas our solver indeed has two modes, add a convenient ability to select from a broader source of solving methods, that makes the program more adaptive to particular user’s tastes or puzzle’s features.

9.2 Implementation

Class based approach perfectly emphasizes on the usability of the function, which also affects the reusability and maintainability of the code. The separation of concerns that supports different ways of solving, verifying, and producing Sudoku puzzles enhances the code readability and understandability.

One potential enhancement could be to increase the number crunching powers of the software, for example, it could be useful for hard puzzles. Even though the algorithm works well for standard-sized puzzles, it might need to be improved or experiments with parallelization might be performed to achieve higher speed and be sure that challenges of the demanding fields are effectively overcome.

9.3 Testing

Extensive testing is part of the Sudoku project procedures, aiming to guarantee its precision and also correctness. The samples provided that can be used for both solving the problem (for testing the solver's correct functioning) and creating the puzzles is good for verifying the solver work in all possible situations. Additionally, edge cases beyond the ones mentioned, like empty puzzles or invalid input, need to be explored as a broader range of scenarios is covered that way. For test purposes we wrote the python file SudokuTesting and it is calling all the three functions used to calculate diagonal sudoku, backtracking and backtracking and constraint propagation. Finally the speed of a moving body in a particular direction is calculated and then the solution is separated.

With the integration of the automated testing frameworks, likened to Unittest or Pytest, the burden of testing the project could be truly alleviated, with a much higher potential to identify defects or regression and eventually stabilize the project's resilience in the long run.

9.4 User Experience

The project is characterized by an easy to use and intuitive UX. The solvers become available as simple function calls allowing the users to perform the relevant tasks very quickly, such as solving puzzles or creating new ones. Customizable characteristics, such as the amount of clues available during the puzzle production, bring a lot of flexibility that increase the user interest.

Adding a GUI (graphical user interface) adds to the solver's usability as more and more users do not have experience in programming and command-line interfaces, which makes it more accessible for a wider range of users. It gives the users an additional functionality which includes interactive features and also visual feedback that would really enhance the general user experience.

9.5 Documentation and Resources

We provide thorough and easy-to-understand documentation to ensure that users make the best use of the SudokuSolver and Generator. In this regard, better guidance concerning the project's usability and accessibility can be obtained by combining usage examples with API references and troubleshooting suggestions, while code comments currently provide only limited guidance.

In addition to providing users with an instant-solving tool, including tutorials and explanations on Sudoku solving approaches would be an excellent way to enhance the user experience.

9.6 Interaction with the Community

An organization set to coordinate the Sudoku project would make it easier for people to collaborate, offer feedback, and become developers. Sustainability of the project can be better solved by providing communication channels in terms of mailing lists, social networks, or forums. These avenues are meant to enhance communication, information sharing as well as idea exchange.

Through open source platforms like GitHub that allow users to update, change, or add features for the betterment and evolution of the entire infrastructure of Sudoku, joint effort can be incited and the strength of the project can grow by leaps and bounds.
(https://ramblings.mcpher.com/recursion/sudoku-generator-and-solver/)

10. References

1. https://www.rd.com/list/printable-sudoku-puzzles/
2. https://www.theguardian.com/media/2005/may/15/pressandpublishing.usnews
3. https://ieeexplore.ieee.org/abstract/document/6779291
4. https://stackoverflow.com/questions/63497248/sudoku-solver-in-python-not-printing-board
5. https://www.geeksforgeeks.org/backtracking-algorithms/
6. https://www.ibm.com/docs/en/icos/20.1.0?topic=optimizer-constraint-propagation#:~:text=over%20this%20variable.-,Constraint%20propagation%20is%20the%20process%20of%20communicating%20the%20domain%20reduction,communicated%20to%20the%20appropriate%20constraints.
7. https://rosettacode.org/wiki/Knuth%27s_algorithm_S
8. https://www.sciencedirect.com/topics/computer-science/stochastic-algorithm#:~:text=Stochastic%20algorithms%20are%20particularly%20interesting,to%20solve%20various%20mathematical%20problems
9. https://github.com/Mercrist/Sudoku-GUI
10. http://www.diva-portal.org/smash/get/diva2:811020/FULLTEXT01.pdf