**Vietnam National University – HCM International University**

**School of Computer Science and Engineering**



**ALGORITHMS AND DATA STRUCTURES**

**PROJECT REPORT**

Semester 2, 2023 – 2024

**Instructor: Vi Chi Thanh, Thai Trung Tin**

**Topic: Sudoku Solver**

Name: Nguyen Nhat Khiem

ID: ITDSIU21091

Github repository: [link](https://github.com/Khim3/Sudoku_Solver_DSA_Project)

Contents

[**CHAPTER 1: INTRODUCTION** 3](#_Toc168933144)

[**Abstract** 3](#_Toc168933145)

[**Overview** 3](#_Toc168933146)

[**Project objectives** 3](#_Toc168933147)

[**Specification** 4](#_Toc168933148)

[**CHAPTER 2: PROJECT TIMELINE** 4](#_Toc168933149)

[**CHAPTER 3: METHODOLOGY** 5](#_Toc168933150)

[**Algorithms and Data Structures application** 5](#_Toc168933151)

[**Project Structure** 7](#_Toc168933152)

[**UML** 10](#_Toc168933153)

[Class Descriptions 11](#_Toc168933154)

[Dependency and Association 13](#_Toc168933155)

[**User Interface** 13](#_Toc168933156)

[**App Implementation** 18](#_Toc168933157)

[**CHAPTER 4: CONCLUSION** 19](#_Toc168933158)

[**Accomplishment** 19](#_Toc168933159)

[**Incomplete problems** 20](#_Toc168933160)

[**Future works** 20](#_Toc168933161)

[**Final words** 21](#_Toc168933162)

[**References** 21](#_Toc168933163)

# **CHAPTER 1: INTRODUCTION**

## **Abstract**

This project presents a Sudoku solver implemented in Python using the Tkinter library for the graphical user interface. The solver utilizes two distinct algorithms: Backtracking and AC3 (Arc Consistency Algorithm 3), demonstrating their effectiveness in solving Sudoku puzzles. The comparison of these two algorithms provides insights into their efficiency and performance in different scenarios.

## **Overview**

Sudoku is a popular puzzle game that requires logical reasoning and has been a subject of interest in the field of computer science and artificial intelligence. This project aims to develop a Sudoku solver that can solve any valid Sudoku puzzles. The solver is implemented in Python, a high-level programming language known for its simplicity and readability. The graphical user interface is developed using Tkinter, Python’s standard GUI package, providing an interactive platform for users to input puzzles and visualize the solving process.

## **Project objectives**

The primary objectives of this project are:

1. To implement a Sudoku solver using the Backtracking algorithm and AC3 algorithm. This involves understanding and applying the principles of Data Structures and Algorithms (DSA) to effectively use these algorithms for problem-solving.
2. To compare the performance and efficiency of the Backtracking and AC3 algorithms in solving Sudoku puzzles. This will provide insights into the strengths and weaknesses of each algorithm, contributing to the broader understanding of DSA.
3. To develop a user-friendly graphical interface using Tkinter that allows users to input puzzles and visualize the solving process. This involves applying principles of User Interface (UI) and User Experience (UX) design to ensure the application is intuitive and engaging.
4. To create an application providing a high-quality user experience (UX) with features such as solving predefined problems with various difficulties, creating random puzzles, and providing solutions using the selected algorithms. This enhances the practical utility of the application for users.
5. To design the application using Object-Oriented Programming (OOP) principles to ensure the code is modular, extensible, and maintainable. This allows for easy addition of new features or modification of existing ones in the future.
6. To deepen understanding of DSA and OOP concepts through the practical implementation of this project. This serves as a valuable learning experience, reinforcing theoretical knowledge with hands-on application.

## **Specification**

The project is implemented in Python, utilizing the Tkinter library for the graphical user interface. The solver includes two algorithms: Backtracking, a brute-force algorithm that tries all possible values until a solution is found or all possibilities are exhausted; and AC3, a constraint satisfaction algorithm that reduces the domain of each variable by removing values that are inconsistent with the constraints.

|  |  |
| --- | --- |
| Tools/Software | Purpose |
| Python 3.10.11 | for running, compiling, debugging Python programs |
| Tkinter 8.6.12 | main Python library for creating UI |

Table 1. Tool specification

# **CHAPTER 2: PROJECT TIMELINE**

|  |  |  |
| --- | --- | --- |
| Stage | Tasks | Week no. |
| Planning | Topic research | 1 |
| Set up timeline |
| Define specififation, requirements |
| Topic confirmation |
| Preparation | Create github repo | 2 |
| Design project structures |
| Get familiar with tkinter and python | 3 - 4 |
| Decide the scopes, aims, and target audients of the projects |
| Implementation | Create the main file with basic UI, add singleton design | 5 |
| Design OOP structure for the project files |
| Modeling the problem as CSP | 6-7 |
| Implement backtracking and AC3 and evaluate with simple puzzles |
| Create input problems with various difficulties (easy, medium, hard) | 8-9 |
| Design the UI appearance |
| Add buttons with functions to the UI | 10 |
| Add custom input feature as well as random problem generator |
| Evaluate and test for potential bugs | 11 |
| Finalize and clean up code | 12 |
| Presentation | Final report and readme | 13-15 |
| Presentation slides |

Table 2. Timeline of the project

# **CHAPTER 3: METHODOLOGY**

## **Algorithms and Data Structures application**

The project employs two primary algorithms for solving Sudoku puzzles: Backtracking and AC3 (Arc Consistency Algorithm 3).

* **Algorithms**
* Backtracking: The Backtracking algorithm is a depth-first search algorithm for finding solutions to constraint satisfaction problems. In the context of Sudoku, it works by assigning numbers to empty cells in a sequential manner and checking if the current assignment leads to a solution. If not, it backtracks and tries the next number. The algorithm continues this process until a solution is found or all possibilities have been exhausted.
* The Recursive\_Backtracking method is the core of the algorithm, which recursively tries all possible values for unassigned variables and backtracks whenever it determines that a variable cannot be assigned a value without violating the constraints.
* The AC3 (Arc Consistency Algorithm 3) is a constraint propagation algorithm used in constraint satisfaction problems to reduce the search space. It works by making each variable arc-consistent with each other variable. A variable is arc-consistent with another variable if each of its admissible values is consistent with some admissible value of the second variable.
* The AC3 method is the core of the algorithm, which iteratively makes each variable arc-consistent with each other variable until the domain of each variable is reduced as much as possible.
* **Data Structures**
* Variables: A list of variables is created, each representing a cell in the Sudoku grid. This is done using the cross function to generate a list of cell identifiers (e.g., ‘A1’, ‘A2’, …, ‘I9’).
* Domains: A dictionary is created where the keys are the variables and the values are the possible numbers that can be assigned to each variable (i.e., the domain of the variable).
* Units and Peers: For each variable, a list of units (i.e., the row, column, and 3x3 square that the variable belongs to) and peers (i.e., the other variables that belong to the same units) are created. These are used to check the constraints of the Sudoku puzzle.
* Constraints: A list of constraints is created, each representing a pair of variables that must be different.
* Queue: Used in the AC3 algorithm to store pairs of variables that need to be made arc consistent. The queue ensures that all necessary pairs are checked in an efficient manner.
* **Comparison of Algorithms**
* Backtracking is a depth-first search algorithm. It’s simple and guaranteed to find a solution if one exists, by exhaustively trying all possibilities. However, it can be slow for complex puzzles because it doesn’t consider the constraints of Sudoku until it completes a path.
* AC3 (Arc Consistency Algorithm 3), on the other hand, is a constraint propagation algorithm. It reduces the domain of each variable by removing values that are inconsistent with the constraints, which can significantly reduce the search space. This makes AC3 more efficient than Backtracking for many puzzles. However, AC3 can’t solve puzzles that require guessing and checking - it only propagates constraints.

|  |  |  |
| --- | --- | --- |
|  | Backtracking | AC3 |
| Time complexity | or |  |
| Space complexity |  |  |
| Time taken | Slow, can be longer for complex problems | Fast, but time taken can vary |
| Completeness | Always | Not always |
| Optimality | Less optimal | More optimal |
| Data Structue | Stack | Queue |
| Applicability | Any csp | Arc consistency greatly trims the search space |

Table 3. Algorithm comparison

## **Project Structure**

A screenshot of a computer

Description automatically generated

Figure 1. Structure of the whole project

The project is structured in a way that ensures clear separation between different components, making it easier to navigate and understand the application’s functionality. The project is organized into several directories and files, each serving a specific purpose within the application.

The `input` directory contains text files with Sudoku puzzles of varying difficulties (Easy.txt, Medium.txt, Hard.txt) and predefined problems (Problem1.txt, Problem2.txt, Problem3.txt) and Custom.txt for sovling puzzle input from the user. The \_\_pycache\_\_ directory stores compiled Python files to improve the loading time of the modules. The output directory stores the output files after solving the Sudoku puzzles, with corresponding names to the input files (e.g., Easy\_output.txt, Medium\_output.txt ...).

In terms of assets, the report of the project is stored in the assets folder. AC3.py implements the AC3 algorithm for solving Sudoku puzzles. Backtrack.py contains the backtracking algorithm used for puzzle-solving. csp.py stands for Constraint Satisfaction Problems, related to the puzzle-solving strategies. sudoku.py is the main script that contains the logic for solving Sudoku puzzles. util.py contains utility functions that support other scripts in the project. README.md provides instructions and explanations about the software. Finally, requirements.txt lists all the dependencies required to run the project.

## **UML**

A screenshot of a computer

Description automatically generated

Figure 2. Project's UML

### Class Descriptions

#### SudokuSolver

* **Functionality**: Manages the entire user interface for the Sudoku solver application.
* **Attributes**:
  + \_instance: Singleton instance of the SudokuSolver.
  + root: Main window for the Tkinter application.
  + grid\_frame: Frame that contains the Sudoku grid.
  + cells: 9x9 grid of Tkinter Entry widgets for user input.
  + selected\_algorithm: StringVar to hold the selected solving algorithm.
  + algoLabel, algoMenu: Label and Combobox for selecting the solving algorithm.
  + selected\_difficulty: StringVar to hold the selected difficulty level.
  + difficultyLabel, difficultyMenu: Label and Combobox for selecting the difficulty level.
  + solveButton, clearButton, validateButton, generateButton: Buttons for solving, clearing, validating, and generating puzzles.
  + timeLabel, time: Label and StringVar for displaying the solving time.
  + timeDisplay: Label to show the solving time.
  + openButton: Button to open a predefined solution.
  + initial\_cells: Set of initially filled cells to avoid overwriting.
* **Methods**:
  + \_\_init\_\_(root): Initializes the GUI components.
  + \_\_new\_\_(cls): Ensures the singleton pattern.
  + get\_instance(cls, root): Returns the singleton instance.
  + create\_widgets(): Creates and arranges the GUI widgets.
  + open\_solution(): Opens a solution from a file.
  + validate(): Validates the current Sudoku grid.
  + solve(): Solves the Sudoku puzzle using the selected algorithm.
  + \_solve(): Internal method for solving the puzzle.
  + generate\_random(): Generates a random Sudoku puzzle.
  + clear(): Clears the Sudoku grid.

#### csp

* **Functionality**: Represents the core structure of the Constraint Satisfaction Problem (CSP) for Sudoku.
* **Attributes**:
  + variables: List of all variables (cells) in the Sudoku grid.
  + domain: Dictionary mapping each variable to its possible values.
  + unitlist: List of all units (rows, columns, boxes) in the grid.
  + units: Dictionary mapping each variable to the units it is part of.
  + peers: Dictionary mapping each variable to its peers.
  + constraints: List of all binary constraints between variables.
  + values: Dictionary of the current values assigned to each variable.
* **Methods**:
  + \_\_init\_\_(domain, grid): Initializes the CSP with a given domain and grid.
  + getDict(grid): Converts a grid string into a dictionary of values.

#### BacktrackingSolver

* **Functionality**: Implements the backtracking search algorithm to solve Sudoku puzzles.
* **Attributes**:
  + csp: The CSP instance that the solver will work on.
  + assignment: Dictionary to keep track of the current variable assignments.
* **Methods**:
  + \_\_init\_\_(csp): Initializes the solver with a CSP instance.
  + Backtracking\_Search(csp\_instance): Initiates the backtracking search.
  + Recursive\_Backtracking(): Recursively performs backtracking search.
  + Inference(inferences, var, value): Makes inferences to reduce domains.
  + Select\_Unassigned\_Variables(): Selects the next variable to assign using the Minimum Remaining Values (MRV) heuristic.
  + isComplete(): Checks if the assignment is complete.
  + isConsistent(var, value): Checks if assigning a value to a variable is consistent with the CSP constraints.
  + write(values): Converts the assignment to a string format.

#### AC3Solver

* **Functionality**: Implements the AC3 (Arc Consistency 3) algorithm to maintain arc consistency in the CSP.
* **Attributes**:
  + csp: The CSP instance that the solver will work on.
* **Methods**:
  + \_\_init\_\_(csp): Initializes the solver with a CSP instance.
  + AC3(): Applies the AC3 algorithm to enforce arc consistency.
  + Revise(Xi, Xj): Revises the domain of a variable based on its neighbor.
  + isConsistent(x, Xi, Xj): Checks if a value assignment is consistent with the CSP constraints.
  + isComplete(): Checks if all variables are assigned.
  + write(values): Converts the assignment to a string format.

#### util

* **Functionality**: Provides utility functions that are commonly used by the CSP and solver classes.
* **Methods**:
  + raiseNotDefined(): Raises an error for unimplemented methods.
  + cross(A, B): Returns the cross product of two sets.

### Dependency and Association

* SudokuSolver depends on csp, BacktrackingSolver, and AC3Solver for solving Sudoku puzzles.
* BacktrackingSolver and AC3Solver depend on csp for the problem structure.
* csp utilizes utility functions from the util class.
* SudokuSolver interacts with tkinter for the GUI, threading for concurrent operations, and random for generating puzzles. These are standard Python modules, so they do not need to be explicitly shown in the UML diagram.

## **User Interface**

* User Interface design

Designing an intuitive and functional user interface was one of the most challenging aspects of this project. To address this, we decided to leverage the simplicity and versatility of the Tkinter library in Python to create our application’s default user interface.

* Main User Interface

Upon launching the application, users are greeted with a main user interface that presents various options, each represented by a button with its unique functionality. By default, pressing the “Solve” button triggers the application to solve a predefined set of 20 basic Sudoku problems using the default algorithm, which is A screenshot of a computer

Description automatically generatedbacktracking.

Figure 3 . The app's interface

* Solution display and algorithm selection

Once the solving process is complete, users can press the “Open Solution” button to display the solution of the recently chosen problems. Users have the flexibility to select the algorithm used for solving, with AC3 being one of the available options.

A screenshot of a computer

Description automatically generated

Figure 4. Option to show the solution for users

* Time Complexity Illustration

To provide users with essential details for algorithm evaluation, we incorporated a time counter that measures the duration of all operations. This feature is particularly useful when illustrating the time complexity of different algorithms, such as the backtracking algorithm when applied to a collection of 15 hard Sudoku puzzles (problem3).

A screenshot of a computer

Description automatically generated

Figure 5. Time taken for solving with Backtracking

* Problem Generation and Customization

A screenshot of a computer

Description automatically generatedFor users interested in solving randomly generated problems, we added a feature that allows the creation of random numbers with varying levels of difficulty: Easy, Medium, and Hard. After setting up these random problems, users can press the “Solve” button to fill the initial problem with the solution. For better visualization, all initial numbers of the problem are displayed in green, while the solution is highlighted in blue.

Figure 6. Solutions for random puzzles

* Custom Problem entry

Finally, one of the key features of our application is the ability for users to enter their own Sudoku problem, check its validity, and select the desired algorithm for the solution through the “Custom” option. This feature provides a high level of customization and user engagement, making our application not only a tool for solving Sudoku puzzles but also a platform for users to test and understand different solving algorithms.

A screenshot of a computer

Description automatically generated

Figure 7. Vadility check notification for custom puzzle

A screenshot of a computer

Description automatically generated

Figure 8.Filled solution for random puzzle

## **App Implementation**

* **Object-Oriented Principles**

The application is structured around the OOP paradigm, which allows for clear encapsulation of data and methods within objects. This encapsulation makes the code more maintainable and scalable. For instance, each Sudoku puzzle is an object with its own data (the puzzle grid) and methods (such as validate and solve).

* **Design pattern:**

The Singleton pattern is used to ensure that only one instance exists throughout the application. This pattern is crucial in our application to avoid creating multiple interface instances, which could lead to inconsistencies and increased memory usage.

* **Backtracking and AC3 Algorithms implemenation**

The Backtracking algorithm is implemented using a recursive function that attempts to place numbers in the Sudoku grid one by one, backtracking whenever it hits a dead end. This algorithm is simple and effective, but it can be slow for complex puzzles.

The AC3 (Arc Consistency 3) algorithm, on the other hand, is a more advanced technique that reduces the problem’s domain size by iteratively enforcing arc consistency. This algorithm is implemented using a queue data structure to keep track of all the arcs that need to be checked for consistency.

* **User Interface**

The user interface, built with the Tkinter library, provides a simple and intuitive way for users to interact with the application. It includes buttons for solving the puzzle, viewing the solution, and generating new puzzles. The interface also allows users to choose the difficulty level for the generated puzzles and select the algorithm (Backtracking or AC3) to be used for solving.

* **Multithreading Utilization**

Multithreading is used to run the Backtracking and AC3 algorithms in parallel, significantly speeding up the solving process. This is achieved by creating a separate thread for each algorithm when the “Solve” button is clicked. The built in module threading is used to manage these threads, ensuring that they are properly started and stopped.

By implementing these features and design patterns, I have created a robust and efficient Sudoku solver application that provides a high-quality user experience. The application not only solves Sudoku puzzles but also serves as a platform for users to understand and compare different solving algorithms. This makes it a valuable tool for both Sudoku enthusiasts and those interested in algorithm design and analysis.

# **CHAPTER 4: CONCLUSION**

## **Accomplishment**

* + Successful implementation of Algorithms:  The project successfully implemented two distinct algorithms, Backtracking and AC3, for solving Sudoku puzzles. This demonstrates a deep understanding of these algorithms and their application in a practical setting. The implementation was done in Python, showcasing proficiency in the language and its application in problem-solving.
  + Performance Comparison: A comprehensive performance comparison was conducted between the Backtracking and AC3 algorithms. This involved solving various Sudoku puzzles and measuring the time and steps taken by each algorithm. The results provided valuable insights into the efficiency and effectiveness of each algorithm, contributing to a broader understanding of their practical applications in problem-solving.
  + Development of User-Friendly GUI: A user-friendly graphical interface was developed using the Tkinter library in Python. This interface allows users to easily input Sudoku puzzles and visualize the solving process in real-time. The GUI is intuitive and engaging, enhancing the user experience and making the application accessible to a wide range of users.
  + Application of OOP Principle and design pattern: The application was designed using Object-Oriented Programming (OOP) principles, ensuring the code is modular, extensible, and maintainable. This design approach allows for easy addition of new features or modification of existing ones in the future. Furthermore, an appropriate design pattern (Singleton) was applied to solve common design problems.

## **Incomplete problems**

* + Unattractive UI design: One of the challenges faced during the project was designing an attractive and intuitive user interface. The aesthetics of the application play a crucial role in user engagement and satisfaction. Balancing functionality and aesthetics to create a visually pleasing yet efficient design was a significant challenge.
  + Advanced Algorithm Implementation: While the project successfully implemented Backtracking and AC3 algorithms, there are other advanced algorithms and techniques like Dancing Links or Stochastic methods that could be explored for solving Sudoku puzzles.

## **Future works**

* + Implement Additional Algorithm: Future iterations of the project will explore the implementation of an additional algorithm for solving Sudoku. This could be a Dancing Link or Hidden Single. The inclusion of another algorithm will not only enhance the functionality of the solver but also provide a broader basis for performance comparison.
  + User Interface Customization: To make the application more engaging and personalized for each user, features allowing users to customize the look and feel of the Sudoku board are planned. This could include different themes, board colors, or even customizable cell sizes.
  + Computer Vision Technology for Solving: One of the most exciting enhancements planned is the incorporation of computer vision technology. This would allow users to input Sudoku puzzles simply by taking a picture of them. The application would then use image processing and optical character recognition (OCR) techniques to extract the puzzle from the image and convert it into a format that the solver can understand.
  + User Guide: To make the application more user-friendly and accessible to a wider audience, a user guide or tutorial is planned. This guide would explain how to use the application, input puzzles, and understand the output. It could also provide some background on how the different algorithms work.

## **Final words**

I would like to express our heartfelt gratitude to our lecturer for their thorough guidance and support throughout our journey. Their expertise and mentorship during our theory, lab sessions have been essential in equipping us with the necessary skills for success. I deeply appreciate the comprehensive and well-documented resources they provided, which acted as a robust base for our learning and project development. We also extend our commendations to me for being hard-working, resourceful, and consistent. I am really eager to continue on this path, gaining further insights and knowledge.

## **References**

1. SÁCH LÊ THÀNH. (n.d.). Retrieved from <https://www.youtube.com/@sachlethanh9853>
2. Welcome to Python.org. (n.d.). Retrieved from <https://www.python.org/>
3. Real Python. (2023). Python GUI Programming With Tkinter. Retrieved from <https://realpython.com/python-gui-tkinter/>
4. Sorting Algorithm Visualizer using Python : Part 1(GUI Tkinter). (2020). Retrieved from <https://www.youtube.com/watch?v=RD1JqhtNgvw&list=PLpp8-k7G_6Y2eVX0WtvsDzkH1PhW2d-Rx>
5. TrungDS10. (n.d.). Introduction-to-Artificial-Intelligent/Sudoku at main · TrungDS10/Introduction-to-Artificial-Intelligent. Retrieved from <https://github.com/TrungDS10/Introduction-to-Artificial-Intelligent/tree/main/Sudoku>