**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](#_Toc169209694)

[**Overview** 3](#_Toc169209695)

[**Project objectives** 3](#_Toc169209696)

[**CHAPTER 2: PROJECT TIMELINE** 3](#_Toc169209697)

[**CHAPTER 3: METHODOLOGY** 4](#_Toc169209698)

[**Algorithms and Data Structures application** 4](#_Toc169209699)

[**Project Structure** 5](#_Toc169209700)

[**UML** 5](#_Toc169209701)

[Class Descriptions 6](#_Toc169209702)

[Dependency and Association 7](#_Toc169209703)

[**User Interface** 7](#_Toc169209704)

[**App Implementation** 11](#_Toc169209705)

[**CHAPTER 4: CONCLUSION** 12](#_Toc169209706)

[**Accomplishment** 12](#_Toc169209707)

[**Incomplete problems** 12](#_Toc169209708)

[**Future works** 12](#_Toc169209709)

[**References** 13](#_Toc169209710)

# **CHAPTER 1: INTRODUCTION**

## **Overview**

This project is a Sudoku solver developed in Python. It uses logical reasoning to solve any valid Sudoku puzzles. The user-friendly interface, built with Python’s Tkinter GUI package, allows users to input puzzles and visualize the solving process interactively.

## **Project objectives**

The primary objectives of this project are:

1. Implement a Sudoku solver using Backtracking and AC3 algorithms.
2. Compare the performance of these algorithms in solving Sudoku puzzles.
3. Develop an intuitive UI using Tkinter for puzzle input and solution visualization.
4. Enhance user experience with features like solving predefined or random puzzles.
5. Deepen understanding of DSA and OOP concepts through practical application.

# **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: The project uses Backtracking and AC3 algorithms for solving Sudoku puzzles. Backtracking assigns numbers to cells sequentially, backtracking when no solution is found. AC3 reduces the search space by making each variable arc-consistent with each other.
* Data Structures: The project uses lists for variables, units, and peers, a dictionary for domains, and a queue for the AC3 algorithm.
* Comparison: Backtracking is simple but can be slow for complex puzzles. AC3 is more efficient as it reduces the search space, but it can’t solve puzzles that require guessing and checking.

|  |  |  |
| --- | --- | --- |
|  | 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 |

16of30

Table 3. Algorithm comparison

## **Project Structure**

The project is organized into directories for inputs, outputs, and assets. The input directory contains Sudoku puzzles, and the output directory stores the solutions. Key files include AC3.py and Backtrack.py for the Sudoku-solving algorithms, csp.py for constraint satisfaction problems, sudoku.py for the main logic, and util.py for utility functions. README.md provides instructions and explanations, and 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 Sudoku solver application's UI.
* Attributes:

\_instance: Singleton instance.

root: Tkinter main window.

grid\_frame, cells: GUI elements for the Sudoku grid.

selected\_algorithm, algoLabel, algoMenu: Components for algorithm selection.

selected\_difficulty, difficultyLabel, difficultyMenu: Components for difficulty selection.

solveButton, clearButton, validateButton, generateButton: Control buttons.

timeLabel, time, timeDisplay: Components for displaying solving time.

openButton: Button to open a predefined solution.

initial\_cells: Cells that are initially filled.

* Methods:

\_\_init\_\_(root), \_\_new\_\_(cls), get\_instance(cls, root): Initialization and singleton pattern.

create\_widgets(): Sets up the GUI.

open\_solution(), validate(), solve(), \_solve(), generate\_random(), clear(): Core functionalities.

#### csp

* Functionality: Represents the CSP for Sudoku.
* Attributes: variables, domain, unitlist, units, peers, constraints, values: Core CSP components.
* Methods:

\_\_init\_\_(domain, grid): Initializes the CSP.

getDict(grid): Converts grid string to dictionary.

#### BacktrackingSolver

* Functionality: Implements backtracking search for solving Sudoku.
* Attributes:

csp, assignment: CSP instance and variable assignments.

* Methods:

\_\_init\_\_(csp): Initializes the solver.

Backtracking\_Search(csp\_instance), Recursive\_Backtracking(): Performs backtracking search.

Inference(inferences, var, value), Select\_Unassigned\_Variables(), isComplete(), isConsistent(var, value), write(values): Search and inference methods.

#### AC3Solver

* Functionality: Implements the AC3 algorithm for arc consistency.
* Attributes:

csp: CSP instance.

* Methods:

\_\_init\_\_(csp): Initializes the solver.

AC3(), Revise(Xi, Xj), isConsistent(x, Xi, Xj), isComplete(), write(values): Arc consistency methods.

#### util

* Functionality: Provides utility functions.
* Methods:

raiseNotDefined(): Error for unimplemented methods.

cross(A, B): Returns cross product of two sets.

### Dependency and Association

* SudokuSolver depends on csp, BacktrackingSolver, and AC3Solver.
* BacktrackingSolver and AC3Solver depend on csp.
* csp uses utilities from util.
* SudokuSolver interacts with Tkinter, threading, and random modules.

## **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 backtracking.

* A screenshot of a computer

  Description automatically generatedSolution display and algorithm selection

Figure 3 . The app's interface

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

For users interested in solving randomly generated problems, Users can choose from Easy, Medium, or Hard levels, and solve them with a click of the “Solve” button. The initial problem numbers are shown in green, and the solution is highlighted in blue.

A screenshot of a computer

Description automatically generated

Figure 6. Solutions for random puzzles

* Custom Problem entry

The “Custom” option in the aplication allows users to input their own Sudoku puzzles, validate them, and choose the solving algorithm. This feature enhances user engagement by offering customization and serves as a learning platform for understanding different Sudoku-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**

* OOP: The app uses Object-Oriented Principles for clear data and method encapsulation, making the code maintainable and scalable.
* Design Pattern: The Singleton pattern ensures only one instance exists, avoiding inconsistencies and increased memory usage.
* Algorithms: Backtracking and AC3 algorithms are implemented for solving Sudoku puzzles. Backtracking uses a recursive function, while AC3 reduces the problem’s domain size by enforcing arc consistency.
* User Interface: The Tkinter library is used to build a simple and intuitive interface, allowing users to solve puzzles, view solutions, generate new puzzles, and choose difficulty levels and solving algorithms.
* Multithreading: Multithreading runs the Backtracking and AC3 algorithms in parallel, speeding up the solving process. Threads are managed using the built-in threading module. This robust and efficient Sudoku solver not only solves puzzles but also helps users understand and compare different solving algorithms.

# **CHAPTER 4: CONCLUSION**

## **Accomplishment**

* + Algorithms: The project successfully implemented Backtracking and AC3 algorithms in Python for solving Sudoku puzzles.
  + Performance: A detailed comparison was conducted between the two algorithms, providing insights into their efficiency and effectiveness.
  + GUI: An intuitive and engaging graphical interface was developed using the Tkinter library, allowing users to input puzzles and visualize the solving process.
  + Design: The application was designed using Object-Oriented Programming principles and the Singleton design pattern, ensuring modularity, extensibility, and maintainability of the code.

## **Incomplete problems**

* UI Design: Balancing aesthetics and functionality for an engaging user interface was a challenge.
* Algorithm Implementation: While Backtracking and AC3 were used, other advanced techniques like Dancing Links or Stochastic methods could be explored for Sudoku solving.

## **Future works**

* + Additional Algorithm: We plan to add another Sudoku-solving algorithm for improved functionality and performance comparison.
  + UI Customization: Future updates will include options for users to personalize the Sudoku board’s appearance.
  + Computer Vision: We aim to incorporate technology that allows users to input puzzles by taking a picture, which the app will process and convert for solving.
  + User Guide: A user-friendly guide is planned to help users navigate the application and understand its output.

## **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>