Solving 4x4 Sudoku Using Tabu Search and GUI Integration in Python

HAMDY MOHAMED—MAHMOUD ABDELRAOUF

May 6, 2025

Abstract

This paper presents a Python-based 4x4 Sudoku solver using the Tabu Search algorithm. The project includes a user-friendly graphical interface where the player inputs the initial values and can request hints to reveal a single cell at a time. The aim is to demonstrate the use of metaheuristic optimization in constraint-based puzzles and how GUI enhances user interaction.

1 Introduction

Sudoku is a logic-based combinatorial number-placement puzzle. In a 4x4 Sudoku, the goal is to fill a 4x4 grid so that each column, each row, and each of the four 2x2 subgrids contain all digits from 1 to 4. Traditional solvers use backtracking or constraint propagation, but in this paper, we explore an alternative metaheuristic approach using the Tabu Search algorithm, which allows for intelligent exploration of the solution space. A graphical user interface (GUI) has been implemented to facilitate user input and visualize the solving process.

2 Methodology

The Tabu Search algorithm is a local search technique that uses memory structures to avoid cycling back to recently visited solutions. In our implementation, the algorithm starts with a randomly filled valid board (only respecting subgrid constraints), then iteratively minimizes the number of conflicts in rows and columns by swapping values. A tabu list prevents reversing recent moves. The GUI was developed using Python's Tkinter library to allow users to input initial values and press a "Hint" button to fill in one correct cell at a time, and there is a button to solve the game and fill all cells .

3 Brief Analysis of Tabu Search

Tabu Search is effective in escaping local optima by maintaining a tabu list of recent moves that are temporarily forbidden. This allows the algorithm to explore non-improving paths when necessary. In our Sudoku solver, each iteration considers all possible swaps in non-fixed cells and selects the one that results in the greatest reduction in conflict, excluding tabu moves unless an aspiration criterion is met. The search continues for a fixed number of iterations or until a solution with zero conflicts is found.

4 Implementation and Technical Results

The project was implemented in Python. The main components include:

- A function to validate Sudoku constraints.
- The Tabu Search solver that works with fixed and non-fixed cells.
- A GUI for user input and interaction, with buttons for hint and solve.

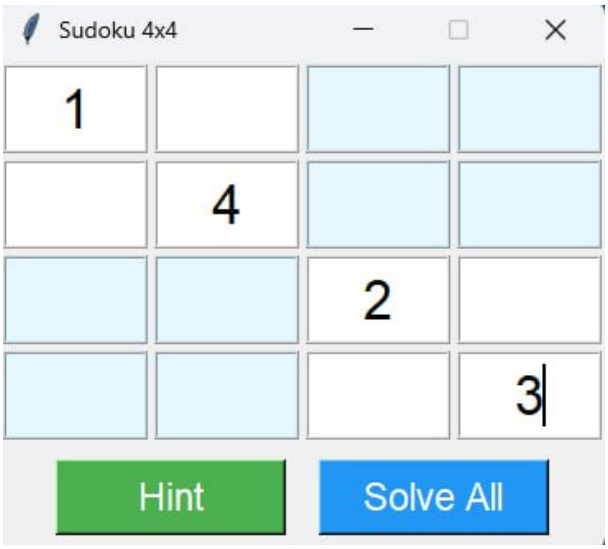

Figure 1: Sudoku GUI with initial user input

Testing on various partially filled 4x4 boards showed that the algorithm can consistently solve the puzzle or give useful hints within 100–200 iterations. The GUI is responsive and updates in real-time when hints are provided.

5 Conclusion

This project demonstrates the feasibility of using Tabu Search for solving small-scale Sudoku puzzles. While it may not be the most efficient algorithm for larger grids, its use in a 4x4 puzzle showed strong results and a good learning experience in metaheuristics. The integration with a GUI made the interaction intuitive and enhanced the user experience.

6 References

- Fred Glover. "Tabu Search—Part I." ORSA Journal on Computing, 1989.
- Sudoku Rules. `https://sudoku.com`
- Tkinter Documentation.
- Python Tabu Search GitHub Examples.