Documentation for the semester project BI-ZUM FIT ČVUT, LS 2022/2023

Student's name: Dmytro Borovko
Username: borovdmy
Title of the semester project: “Sudoku na 100%”

Assignment of the semester project:

Let's consider the well-known puzzle game Sudoku, where we need to fill in the numbers 1 to 9 into a 9x9 grid, following rules:

- Each 3×3 block can only contain numbers from 1 to 9.
- Each vertical column can only contain numbers from 1 to 9.
- Each horizontal row can only contain numbers from 1 to 9.
- Each number in the 3×3 block, vertical column or horizontal row can be used only once.

The task is to solve the Sudoku puzzle using artificial intelligence techniques, but in a way that is as fast as possible and guarantees a correct solution. The solving algorithm should always terminate and provide the correct solution.

General view of the program:

When my program is launched, a window opens immediately, displaying a 9x9 grid where the user can input the Sudoku puzzle they want to solve. It is important to note that if the user incorrectly enters the Sudoku conditions, the program will not allow it. For example, if you try to enter two identical digits in the same row, the program will not accept the second digit. This rule also helps in verifying the correct functioning of the program because it applies to the solution provided by the AI as well. The AI is not allowed to violate the rules when filling in the numbers.

Once you have filled in the Sudoku puzzle you want to solve, you have the option to choose how the AI will solve the problem (the methods will be introduced below) and whether you want to visualize the process or simply get the solution. If you choose the option without visualization, the program will display information about the time taken by the AI to solve the Sudoku puzzle. Additionally, the program's interface includes two buttons: "Solve" to solve the puzzle and "Reset" to clear the grid.

It's worth noting that the AI can detect whether the Sudoku puzzle is solvable or not, and if it's not possible to solve, it will inform you accordingly.

Brief analysis of the problem:

In terms of complexity, Sudoku is classified as an NP-complete problem, which means that it belongs to the class of problems that are both in the class NP (nondeterministic polynomial time) and NP-hard. This classification implies that Sudoku is challenging to solve and that finding an optimal solution (if one exists) can be computationally expensive.

Here is where artificial intelligence comes to our aid. AI algorithms can be used for Sudoku solving. AI algorithms provide a powerful approach to tackle Sudoku efficiently by leveraging advanced techniques in search, constraint satisfaction, and optimization.

Methods selection:

There are three methods implemented in my program: Simple backtracking (brute force), CSP algorithm and SAT solver.

Description of the method's application to the given problem:

Backtracking: This is a simple and straightforward method that systematically tries all possible digit assignments until a valid solution is found. It explores the solution space through a depth-first search, backtracking when an invalid assignment is made.

Constraint Satisfaction Algorithm: Sudoku can be formulated as a constraint satisfaction problem (CSP), and algorithms like backtracking with constraint propagation (AC-3 in my implementation) and local search can be applied. These algorithms use intelligent search strategies and heuristics (there are three different heuristics in my program, which help me to increase efficiency) to explore the solution space efficiently while maintaining consistency with the Sudoku constraints.

SAT Solver: Sudoku can be encoded as a Boolean satisfiability problem (SAT) and solved using SAT solvers. SAT solvers use efficient algorithms and heuristics to search for a satisfying assignment of Boolean variables. Encoding Sudoku as a SAT problem allows leveraging the power of advanced SAT solvers to efficiently find solutions for complex Sudoku variants. The main idea of working in this paradigm is that you transfer the solution of a problem to an already implemented program. That is, if someone creates a more efficient SAT-solver, then the efficiency of your program will increase, even though you don't even have to change anything. There is PicoSAT solver in my program from python library pycosat.

Implementation:

For solving Sudoku I represent the Sudoku puzzle as a CSP (triple (X,D,C), where X - finite set of variables, D - finite domain (of values) for variables, C – set of constraints over X) using dictionary data structure. After that I apply constraint propagation, such as the "arc-consistency" algorithm, to reduce the domains of variables based on the current constraints, and optimal backtracking search with three different heuristics (minimum remaining values heuristic, least-constraining values heuristic and degree heuristic) for selecting the unassigned variable and suitable domain for this variable.

The main idea of my implementation of the SAT paradigm is to encode the Sudoku puzzle as a Boolean satisfiability (SAT) problem using logarithmic encoding. This is achieved by representing the Sudoku grid as a set of variables and creating clauses that enforce the Sudoku rules. After encoding the Sudoku puzzle as SAT clauses, the pycosat.solve() function is used to solve the SAT problem. If a solution is found, it is extracted and returned as the solved Sudoku grid. If the problem is unsatisfiable (no solution exists), it means the Sudoku puzzle is invalid and cannot be solved.