**AI LAB 4&5**

To represent the Sudoku puzzle as a **Constraint Satisfaction Problem (CSP)**, the following components were established:

1. **Variables**: Each cell in the 9x9 Sudoku grid is treated as a variable, named according to its row and column (e.g., 'A1' for the cell in the first row and first column). Thus, we have 81 variables representing all cells from 'A1' to 'I9'.
2. **Domains**: Initially, each variable (cell) can hold any value from 1 to 9. If a cell’s value is predefined in the puzzle, its domain is restricted to that single value. This approach reduces the complexity and guides the constraint satisfaction process by limiting possibilities upfront.
3. **Constraints**: In Sudoku, each number 1-9 must be unique within each row, column, and 3x3 sub-grid. This requirement is represented using **constraints** among the cells:
   * **Rows**: Cells in each row must have unique values.
   * **Columns**: Cells in each column must have unique values.
   * **3x3 Sub-grids**: Cells within each 3x3 sub-grid must also be unique.
4. **Peers and Units**:
   * **Units**: Each cell is associated with three units (its row, column, and 3x3 grid). For instance, cell 'B2' is part of the row 'B', column '2', and the upper-left 3x3 sub-grid.
   * **Peers**: A cell’s peers are the cells in its units (excluding itself). This structure facilitates the checking of constraints, as each cell needs to be consistent with only its peers.

**Design Options Considered**

1. **Backtracking with Forward Checking (Inference)**: This approach keeps track of the remaining valid options for each variable as assignments are made. Forward checking reduces potential conflicts early by removing invalid values from neighboring cells.
2. **Arc Consistency (AC-3 Algorithm)**: AC-3 was chosen as an additional constraint propagation method. It iteratively enforces arc consistency by ensuring that every possible value for a variable has corresponding valid values in its peers. This further reduces the domain size for each variable before beginning the backtracking search, potentially lowering the search space significantly.

**Implementation Choice**

The chosen design relies on a combination of **backtracking search** with **constraint propagation using AC-3**. This combination allows efficient handling of both easy and more complex Sudoku puzzles:

* **Backtracking** provides a systematic way of exploring potential solutions.
* **Constraint propagation (AC-3)** minimizes the domain of each variable early in the process, which reduces the need for backtracking and leads to faster solutions.

This combination of methods is well-suited for Sudoku because it balances flexibility with efficiency, accommodating both straightforward puzzles and those with fewer initial constraints.

**A few thoughts about the assignment**

Unfortunately, the assignment took place during the midterm so I couldn’t dig deep into these interesting concepts. However, I realized that not all the algorithms can outsmart a human brain, I tried to solve 1 solution from the magic tour comparing with the backtracking search and I won (but with AC3 algorithm is a different story :))). I enjoy the time I spent to fix the bug and the only difficulty I met is to imagine how the sudoku rows and cols work on paper and apply it to the array in Python. I spent for about 3 hours on Saturdays and Sundays to work with the assignment so it takes me about half a day to finish everything including this report