CS 4820/5820 – Homework 2 (Fall 2025)

Constraint Satisfaction and Metaheuristic Optimization (Total: 12.5 pts)

Instructions:

- Write clear, well-commented code (Python, Java, or another approved language). Do not use specialized CSP or optimization libraries.
- Submit a single PDF report (AAAI format) along with all code files. The report must include your problem formulation, algorithmic design, results, tables, plots, and discussion.
- All code must be your own. No late submissions.
- Collaboration is not allowed on homework assignments.

Part A: Sudoku as a Constraint Satisfaction Problem (4.0 pts)

Sudoku is a 9×9 puzzle divided into nine 3×3 subgrids. Each cell must contain a digit from 1–9 so that:

- Each row contains all digits 1–9 exactly once.
- Each column contains all digits 1–9 exactly once.
- Each 3×3 subgrid contains all digits 1–9 exactly once.

	7			2			4	6
	6					8	9	
2			8			7	1	5
	8	4		9	7			
7	1						5	9
			1	3		4	8	
6	9	7			2			8
	5	8					6	
4	3			8			7	

Figure 1: Example Sudoku puzzle and its 3x3 subgrid division.

Tasks:

a) Formulate Sudoku as a CSP:

- Variables: 81 cells (A1–I9)
- Domain: $\{1, 2, \ldots, 9\}$ for each unfilled cell
- Constraints: 27 `Alldiff` constraints (9 rows, 9 columns, 9 subgrids)

b) Implement a **Backtracking Search** with MRV, Degree heuristic, and LCV ordering.

c) Extend your solver with **Forward Checking**.

d) Further extend it with **Constraint Propagation using AC-3**. Apply AC-3 after every assignment.

e) Solve at least 3 Sudoku puzzles (easy, medium, hard) and report:

- Number of nodes expanded
- Number of backtracks
- Runtime for: *(i)* Backtracking only, *(ii)* Backtracking + Forward Checking, *(iii)* Backtracking + Forward Checking + AC-3.

Hint: *AC-3 (Arc Consistency 3) makes every directed arc $X \to Y$ consistent: for each value x in X's domain, there exists some value y in Y's*

domain that satisfies the constraint. If not, remove x. When a domain loses values, re-queue its neighbors. AC-3 repeatedly enforces this until no more values are pruned.

Part B: Minimum Conflicts Heuristic (3.0 pts)

Implement the **Minimum Conflicts algorithm** for the n-Queens problem. The goal is to place n queens on an $n \times n$ chessboard so that no two queens attack each other.

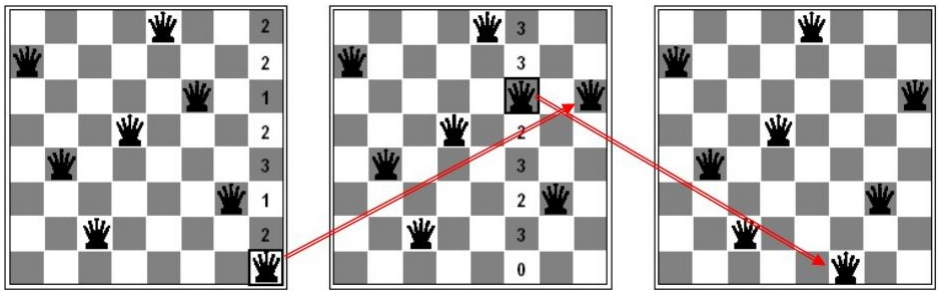

Figure 2: Example configuration illustrating conflicts between queens on the same diagonal.

Tasks:

- Implement Minimum Conflicts for $n = 8, 16, 25$.
- For each n, perform at least 5 runs with random initial configurations.
- Report the average number of steps to reach a solution and average runtime.
- Discuss why Minimum Conflicts performs well for large n despite being a local search.

Hint: *At each iteration, select a conflicted variable and assign it a value that minimizes the number of conflicts. This approach can quickly escape local minima in structured problems like n-Queens.*

Part C: Metaheuristic Optimization (5.5 pts)

This section explores global optimization using metaheuristic algorithms such as Particle Swarm Optimization (PSO), Differential Evolution (DE), and Ant Colony Optimization (ACO).

Part C1: Benchmark Function Optimization (3.0 pts)

Implement and evaluate the performance of at least one of the above algorithms on two continuous benchmark functions:

$$\text{Rastrigin:} \quad f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$$

$$\text{Rosenbrock:} \quad f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

Tasks:

- Implement at least one of PSO, DE, or ACO and run it for a minimum of 3 independent trials on each function.
- Record and present: convergence curves, best fitness values obtained, and total runtime for each trial.
- Experiment with multiple parameter settings (e.g., population size, inertia weight, learning factors for PSO; mutation and crossover rates for DE; pheromone evaporation and heuristic influence for ACO). Report how these choices affect convergence behavior and final performance on both benchmark functions.
- Include comparative plots illustrating convergence speed and final accuracy across parameter configurations.
- Provide a short discussion summarizing which algorithmic choices led to better optimization performance and why.

Part C2: Sudoku via Metaheuristics (2.5 pts)

Use one of your three metaheuristic algorithms (PSO, DE, or ACO) to solve Sudoku as an optimization problem.

Problem setup:

- Represent Sudoku cells as variables, and define a fitness function that counts total constraint violations (row, column, subgrid duplicates).
- Implement your chosen method to minimize this violation count.
- Present and justify your choice of algorithm and its parametter setting.
- Run 3 independent trials on one Sudoku puzzle and report final fitness (number of violations) and runtime.
- Discuss algorithm suitability for discrete problems like Sudoku.

Note for Students: This assignment builds on lectures 5–7 covering Constraint Satisfaction Problems, Minimum Conflicts, and Search Optimization (PSO, DE, ACO). Include figures, tables, and concise reasoning throughout your report.