# CT421: Artificial Intelligence
# Project 1: Genetic Algorithm for Exam Timetabling

January 28, 2026

## 1 Overview

As introduced in class, you will implement a genetic algorithm to solve the exam timetabling problem. This is a well known constraint satisfaction and optimisation problem with many real-world applications.

### 1.1 Learning Objectives

- Implement a genetic algorithm
- Design appropriate chromosome representations for scheduling problems
- Handle hard and soft constraints in optimisation
- Analyze algorithm performance and parameter sensitivity
- Explore the ability of the algorithm to maintain diversity in the search.

## 2 Problem Definition

### 2.1 Input

Given:

- $N$ exams to be scheduled
- $K$ available time slots $(1 \dots K)$
- $M$ students
- An enrollment matrix $E$ where $E[i][j] = 1$ if student $i$ is enrolled in exam $j$, and 0 otherwise

Note that by varying K, M, N and the values in the enrollment matrix we can vary the difficulty of the problem.

### 2.2 Objective

Find an assignment of exams to time slots that satisfies all hard constraints and minimizes soft constraint violations.

### 2.3 Constraints

**Hard Constraint (must be satisfied):**

- No student may have two exams scheduled in the same time slot

**Soft Constraint (minimize): Examples:**

- Total number of consecutive exams for students should be minimised.

## 3 Input File Format

Test instances are provided in plain text format:

```
N K M
e_00 e_01 e_02 ... e_0(N-1)
e_10 e_11 e_12 ... e_1(N-1)
...
e_(M-1)0 e_(M-1)1 ... e_(M-1)(N-1)
```

Where:

- First line: $N$ (number of exams), $K$ (number of slots), $M$ (number of students)
- Following $M$ lines: enrollment matrix (1 if student takes exam, 0 otherwise)

### 3.1 Example Instance: “tinyexample.txt”

```
4 3 5
1 1 0 0
0 1 1 0
0 0 1 1
1 0 0 1
0 1 0 1
```

This represents:

- 4 exams, 3 time slots, 5 students
- Student 0: enrolled in exams {0, 1}
- Student 1: enrolled in exams {1, 2}
- Student 2: enrolled in exams {2, 3}
- Student 3: enrolled in exams {0, 3}
- Student 4: enrolled in exams {1, 3}

# 4 Implementation Requirements

You must implement the following components:

## 4.1 Suggested Functions

1. **read_instance(filename)** – Parse input file
2. **initialize_population(pop_size)** – Create random initial solutions
3. **evaluate_fitness(solution)** – Calculate fitness value
4. **select_parents(population)** – Choose parents for reproduction
5. **crossover(parent1, parent2)** – Create offspring from two parents
6. **mutate(solution)** – Apply random changes to a solution
7. **run_ga()** – Main genetic algorithm loop

## 4.2 Genetic Algorithm Parameters

You should experiment with different parameter values. Suggested starting values:

- Population size: 100
- Number of generations: 500
- Crossover rate: 0.8
- Mutation rate: 0.05
- Selection method: Tournament selection (size 3)

# 5 Output Requirements

Your program should output:

1. The best solution found (array of time slot assignments)
2. The fitness of the best solution
3. Number of hard constraint violations (should be 0 for valid solutions)
4. Measure of soft constraints violated
5. A plot showing fitness over generations

# 6 Tips and Suggestions

- Start with the small instances to debug your implementation
- Plot fitness over generations to ensure your GA is improving
- If your GA gets stuck, try increasing mutation rate or population size
- Consider implementing elitism (keeping the best solution across generations)
- I will upload some test cases later.