CS211FZ: Data Structures and Algorithms II

Genetic Algorithms (again)

LECTURER: CHRIS ROADKNIGHT

CHRIS.ROADKNIGHT@MU.IE

Other Business

Dynamic Programming

Labs

Attendance

Exam

Quiz

Genetic Algorithms

Definition: Genetic algorithms are search techniques inspired by natural evolution to find optimal solutions to complex problems.

Key Idea: Mimic biological processes like selection, crossover, and mutation to evolve solutions over generations.

How It Works: Start with a population of possible solutions, evaluate them, and iteratively improve them.

Applications: Used in optimization, machine learning, scheduling, and design (e.g., engineering, AI).

Process

Population → Evaluation → Evolution → Solution.

Population of possible solution codes

Test all members of the population

Use the ‘best’ solutions to evolve new hybrid solutions

Repeat until satisfactory solution

Core Components of Genetic Algorithms

Population: A set of candidate solutions (e.g., numbers, designs, or schedules).

Fitness Function: Measures how good each solution is (higher fitness = better solution).

Genes: Each solution is encoded as a string (like DNA), often as numbers or binary.

Generations: Solutions evolve over multiple cycles, improving each time.

Genotype	Fitness
1001001011100101010100101	85%
0001010001001001000001011110	75%
1001001000111111010111001111	90%

How Genetic Algorithm's work

Step 1: Initialize: Create a random population of solutions.

Step 2: Evaluate: Use the fitness function to score each solution.

Step 3: Select: Choose the best solutions (parents) based on fitness (e.g., roulette wheel selection, tournament….).

Step 4: Evolve: Create new solutions (offspring) via **crossover** and **mutation**.

Step 5: Repeat: Replace old population, evaluate, and continue until a 'good enough' solution is found.

Key Operations in Evolution

Selection: Pick solutions with higher fitness to reproduce (e.g., best schedules or designs).

Crossover: Combine two parent solutions to create offspring (e.g., mix parts of two strings: "101|10" + "011|01" → "10101", "01110").

Mutation: Randomly tweak offspring to add diversity (e.g., flip a bit: "10101" → "10111").

Purpose: Balance exploration (new solutions) and exploitation (improving good solutions).

Crossover

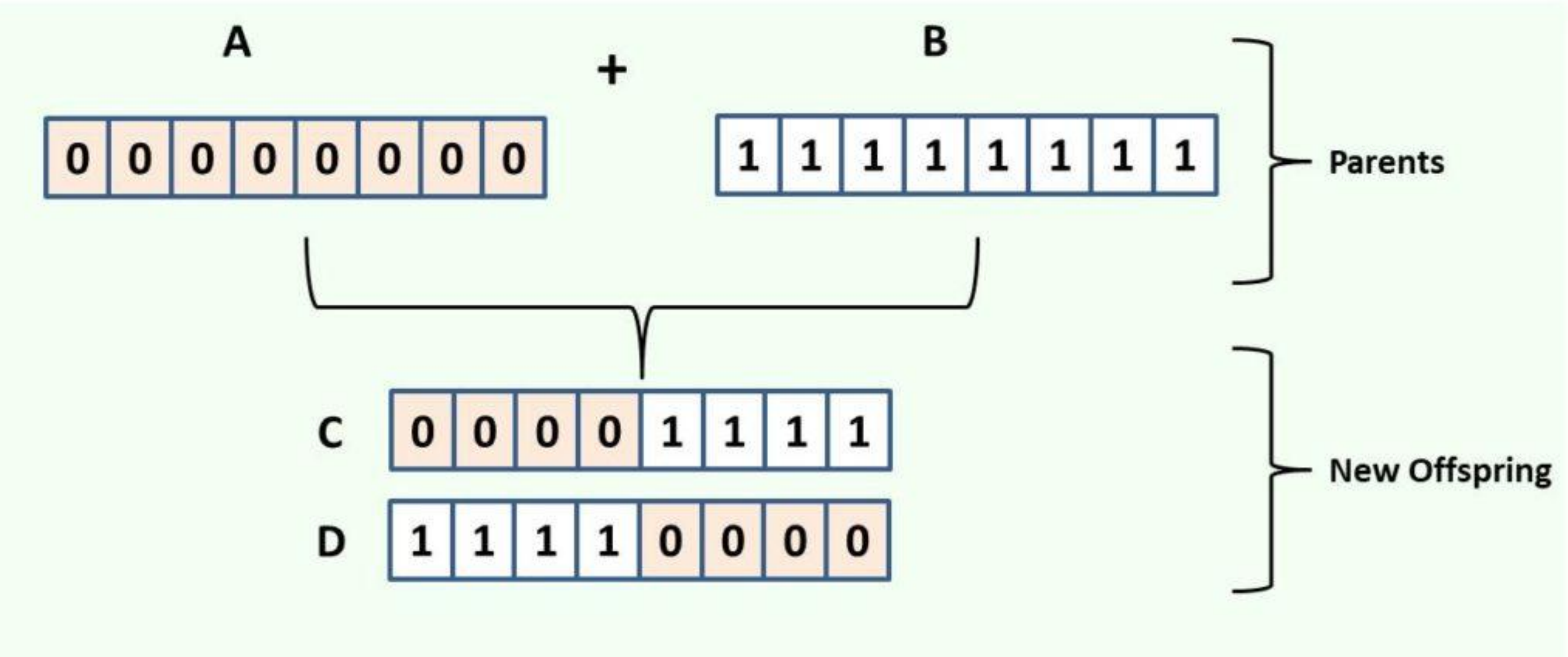
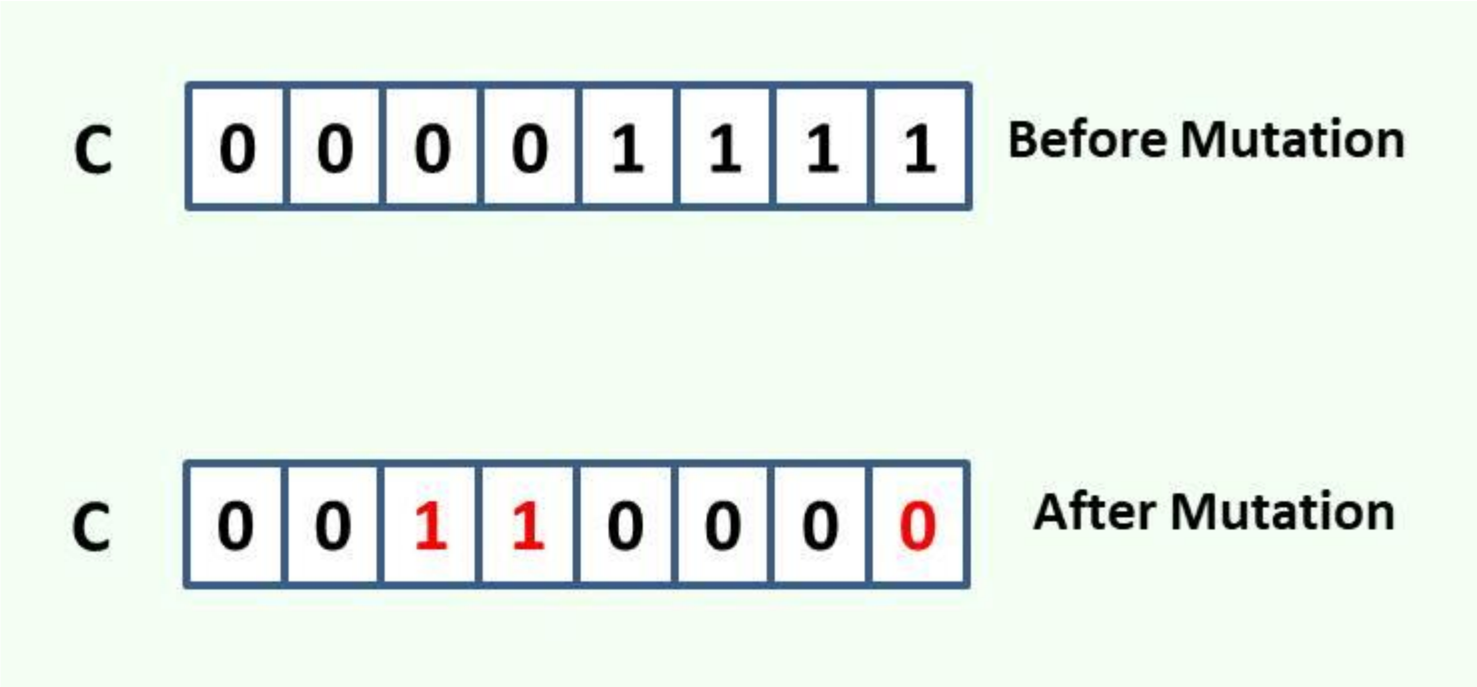
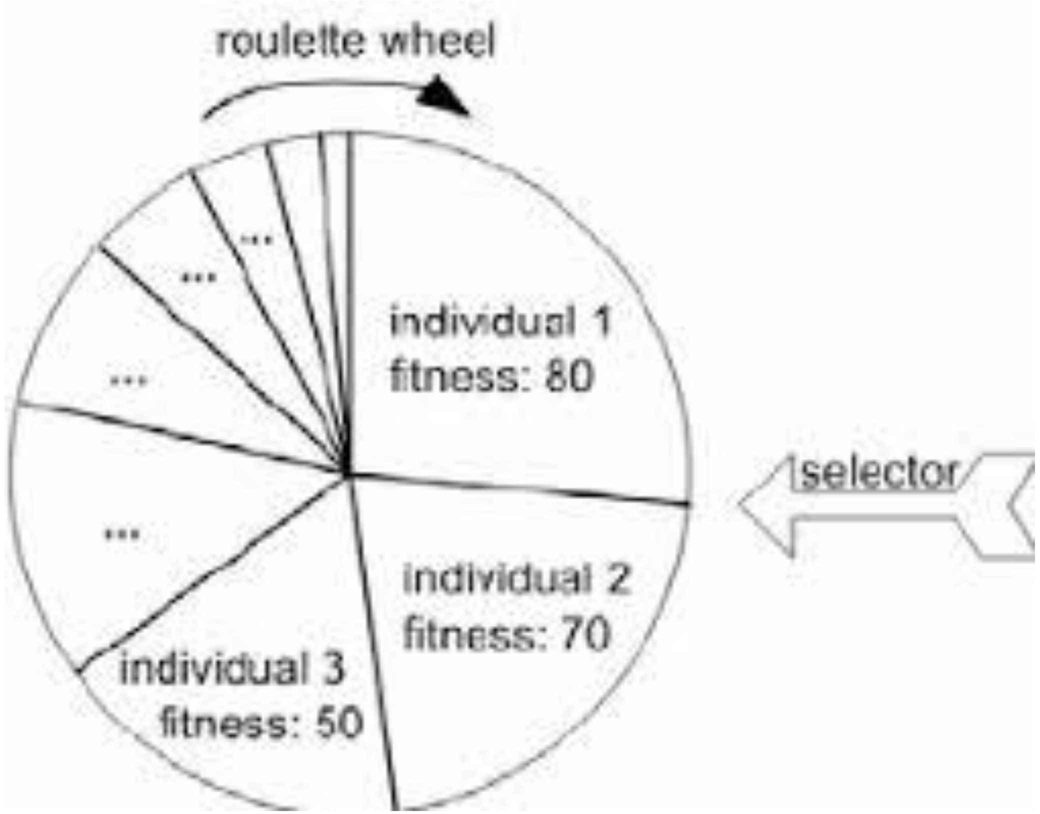


Illustration of Crossover

Mutation



Selection – Biased roulette wheel



Selection – tournament selection

Randomly/biased roulette wheel select 2 candidates.

The highest performing candidate is chosen

Randomly/biased roulette wheel select 2 candidates.

The highest performing candidate is chosen

These two ‘tournament’ winners become the parents

Strengths vs Weaknesses

Strengths:

Solve complex problems where traditional methods fail (e.g., non-linear, multi-variable).

Find near-optimal solutions in large search spaces.

More general than many methods, works with various data types and problems.

Limitations:

May be slow for simple problems.

Requires careful tuning (e.g., mutation rate, population size).

Fitness function – student timetable scheduling

The fitness function may just be the evaluation of a function or may be a set of hard or soft constraints.

Eg. If using GAs for scheduling.

Hard constraints

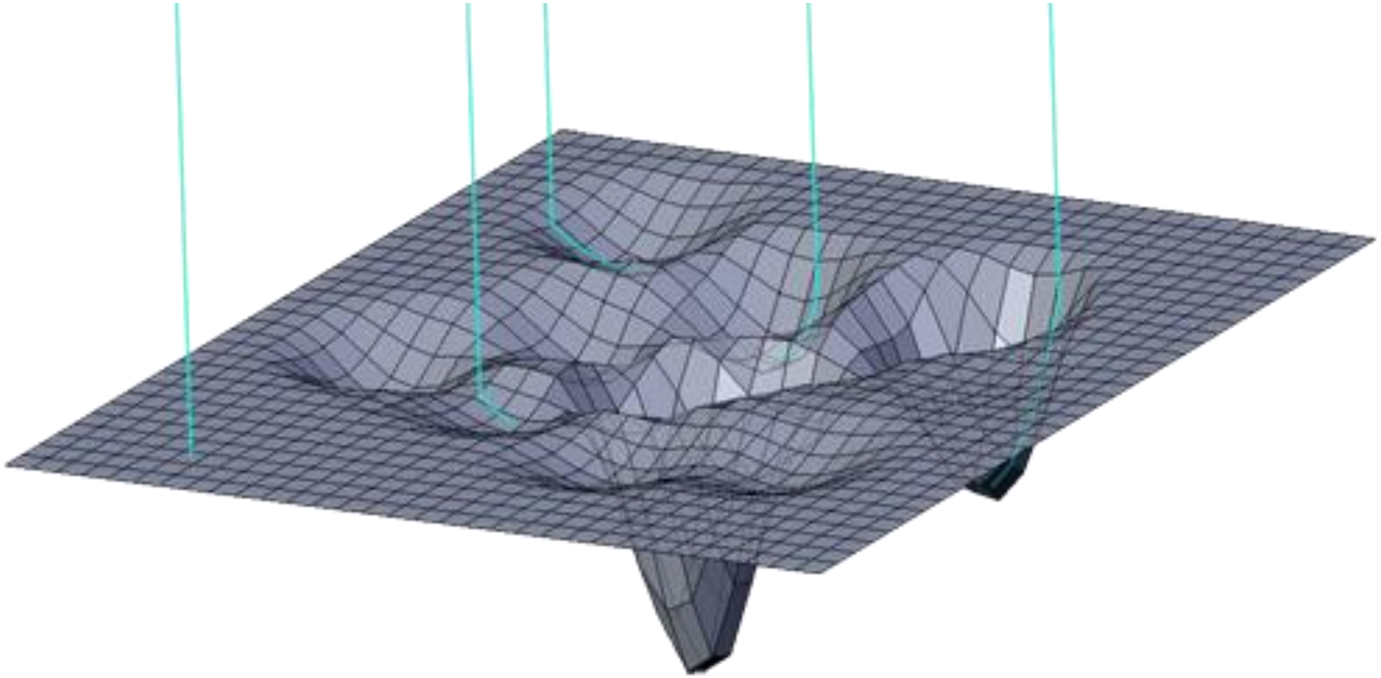
- Student cannot be in two places at the same time (penalty = infinity)
- Impossible to get between some classes fast enough (penalty = infinity)
- Each room has a maximum number of students (penalty = infinity)

Soft Constraints

Sunday afternoons labs should be discouraged (+1000)

Students should have first and last timeslots in one day (+500)

Distance between lectures should be minimised (fitness-distance between rooms in Metres)



Worked example

Decimal	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111

Gray Coding

For many algorithms where the binary number representation is important, Gray coding is used.

Similar numbers have similar representation

Only changing 1 bit at a time