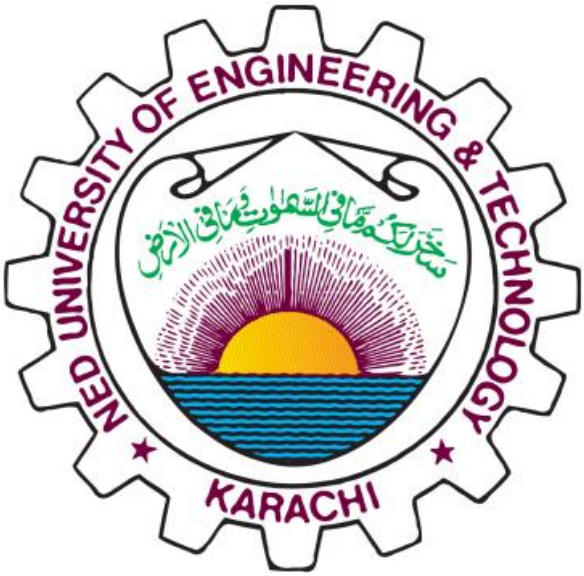
NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY



ARTIFICIAL INTELLIGENCE

OEL REPORT

GROUP MEMBERS:

Iqra Jawad Ahmad	CS-22034
Hiba Fatima	CS-22103
Areesha	CS-22110

SUBMITTED TO:

Ms. Hameeza Ahmed

Table of Contents

S. No.	Description	Pg. No.
1	Introduction	3
2	Problem definition	3
3	Binary String Matching GA overview	3
4	Example	4
5	Novel Simulation	6
6	Application	6
7	Conclusion	6

Introduction

Genetic Algorithms (GAs) are a class of optimization algorithms inspired by the principles of natural selection and genetics. They are widely used to solve complex optimization and search problems by iteratively improving a population of candidate solutions through biologically inspired operations: selection, crossover, and mutation.

In this project, we implemented a GA (**Binary String Matching Genetic Algorithm)**, where the goal is to evolve a population of binary strings to match a specific target binary sequence. The project provides a hands-on simulation of how GAs operate and demonstrates their application in solving a well-defined optimization problem.

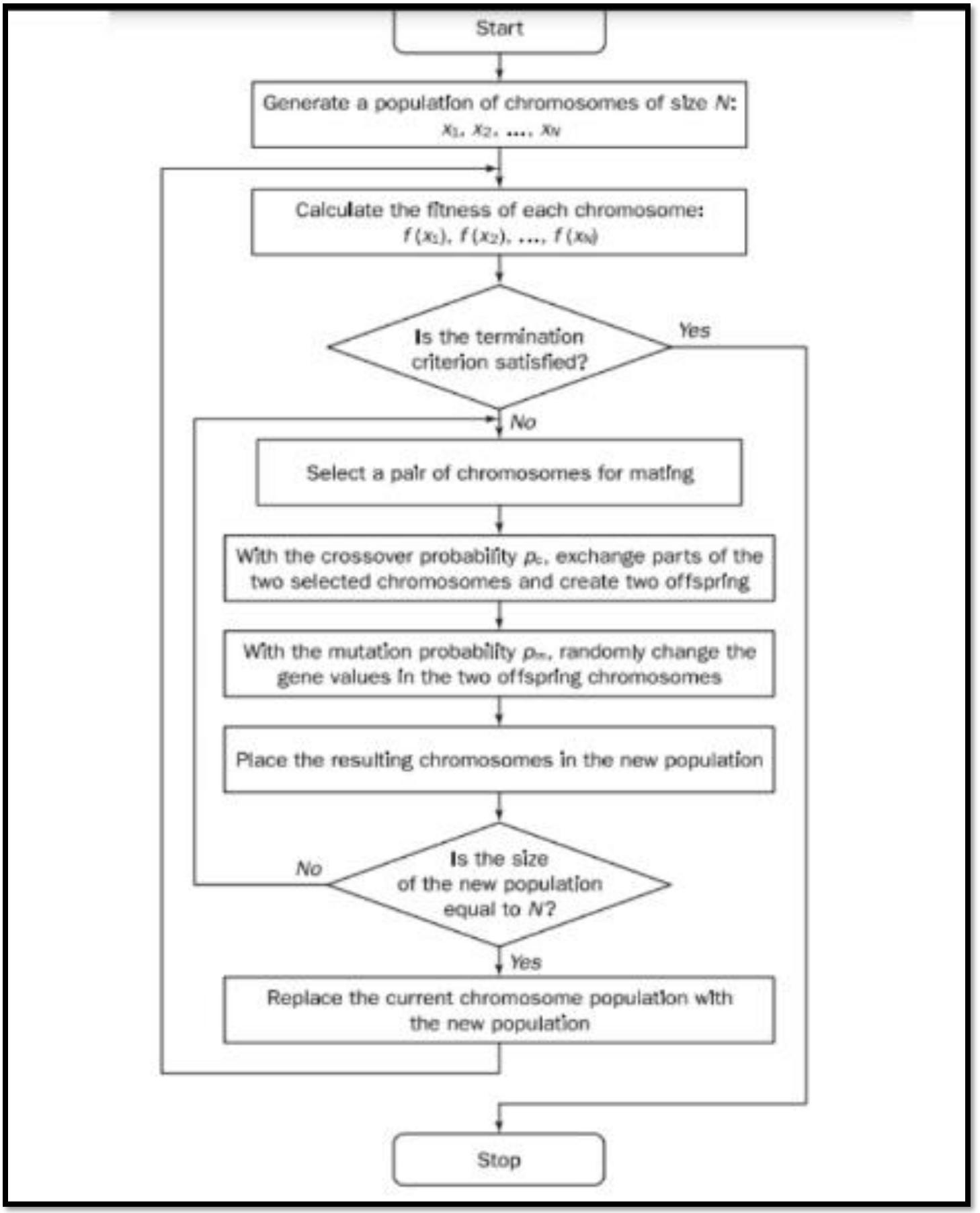
Problem Definition

Given a target binary sequence (e.g., "111111"), the goal is to evolve a binary string (e.g., "101010") that matches target sequence. The GA starts with a random population of binary strings and uses genetic operations (selection, crossover, mutation) to evolve the population over multiple generations. The objective is to reach the target sequence.

Binary String Matching Genetic Algorithm Overview

The Genetic Algorithm follows these steps:

1. **Initialization:**
    - Start with a user-defined initial binary string.
    - Generate a population of random binary strings of size 5 of the same length as the target sequence.
2. **Fitness Evaluation:**
    - Measure the fitness of each individual by comparing it to the target sequence. The fitness is the number of bits that match the target.
3. **Selection:**
    - Select the two fittest individuals as parents for the next generation.
4. **Crossover:**
    - Combine genetic material from the parents to produce offspring. This is achieved by splitting the parents at a random crossover point and exchanging parts of their genetic material. The crossover probability pc is equals to 0.9.
5. **Mutation:**
    - Introduce random changes to the offspring's genes to maintain genetic diversity and avoid local optima. The mutation probability pm is equals to 0.01.
6. **Replacement:**
    - Replace the old population with the new population, ensuring it maintains a fixed size.
7. **Termination:**
    - Repeat the process for a fixed number of generations or until the target sequence is reached.


Start
Generate a population of chromosomes of size N:
x1, x2, ..., xN
Calculate the fitness of each chromosome:
f(x1), f(x2), ..., f(xN)
Is the termination criterion satisfied?
Yes
No
Select a pair of chromosomes for mating
With the crossover probability pc, exchange parts of the two selected chromosomes and create two offspring
With the mutation probability pm, randomly change the gene values in the two offspring chromosomes
Place the resulting chromosomes in the new population
Is the size of the new population equal to N?
No
Yes
Replace the current chromosome population with the new population
Stop

Example

```
GENETIC ALGORITHM FOR BINARY STRING MATCHING
Problem Solved: Evolving a population of binary strings to match a specific target sequence.
Enter the target binary sequence (e.g., 10101010): 110011
Enter the initial binary string (must be same length as target): 010101

Generation 0
Initial Population:
Chromosome: 010101 | Fitness: 3 | Fitness Ratio: 50.00%
Chromosome: 000111 | Fitness: 3 | Fitness Ratio: 50.00%
Chromosome: 001111 | Fitness: 2 | Fitness Ratio: 33.33%
Chromosome: 111010 | Fitness: 4 | Fitness Ratio: 66.67%
Chromosome: 100100 | Fitness: 2 | Fitness Ratio: 33.33%

Selected Best Parents:
Parent 1: 111010 | Fitness: 4
Parent 2: 010101 | Fitness: 3
Applying crossover...

Offspring after crossover:
Crossover offspring: 111011
Crossover offspring: 010100
Crossover offspring: 111101
Crossover offspring: 010010
Crossover offspring: 111101
Crossover offspring: 010010
```

```
Crossover offspring: 111010
Crossover offspring: 010011
Crossover offspring: 110010
Crossover offspring: 011011

Offspring after mutation:
Mutated offspring: 111010
Mutated offspring: 010011
Mutated offspring: 111010
Mutated offspring: 010011
Mutated offspring: 110010
Mutated offspring: 011011

Passing to Generation 2

Generation 2
New Population:
Chromosome: 111010 | Fitness: 4 | Fitness Ratio: 66.67%
Chromosome: 010011 | Fitness: 5 | Fitness Ratio: 83.33%
Chromosome: 111010 | Fitness: 4 | Fitness Ratio: 66.67%
Chromosome: 010011 | Fitness: 5 | Fitness Ratio: 83.33%
Chromosome: 110010 | Fitness: 5 | Fitness Ratio: 83.33%

Selected Best Parents:
Parent 1: 010011 | Fitness: 5
Parent 2: 010011 | Fitness: 5
Applying crossover...
```

```
Offspring after mutation:
Mutated offspring: 111011
Mutated offspring: 010100
Mutated offspring: 111101
Mutated offspring: 010010
Mutated offspring: 111101
Mutated offspring: 010010

Passing to Generation 1

Generation 1
New Population:
Chromosome: 111011 | Fitness: 5 | Fitness Ratio: 83.33%
Chromosome: 010100 | Fitness: 2 | Fitness Ratio: 33.33%
Chromosome: 111101 | Fitness: 3 | Fitness Ratio: 50.00%
Chromosome: 010010 | Fitness: 4 | Fitness Ratio: 66.67%
Chromosome: 111101 | Fitness: 3 | Fitness Ratio: 50.00%

Selected Best Parents:
Parent 1: 111011 | Fitness: 5
Parent 2: 010010 | Fitness: 4
Applying crossover...

Offspring after crossover:
Crossover offspring: 111010
Crossover offspring: 010011
```

```
Selected Best Parents:
Parent 1: 010011 | Fitness: 5
Parent 2: 010011 | Fitness: 5
Applying crossover...

Offspring after crossover:
Crossover offspring: 010011
Crossover offspring: 010011
Crossover offspring: 010011
Crossover offspring: 010011
Crossover offspring: 010011
Crossover offspring: 010011

Offspring after mutation:
Mutated offspring: 010011
Mutated offspring: 010011
Mutated offspring: 110011
Mutated offspring: 010011
Mutated offspring: 010011
Mutated offspring: 010011

Target sequence reached!

Process finished with exit code 0
```

Novel simulation:

- **User-Driven Inputs:** Enables users to customize the target sequence and initial string for flexible optimization.
- **Fitness Feedback Mechanism:** Displays fitness scores and percentages to track progress toward the target.
- **Dynamic Population Initialization:** Combines user-defined initial strings with random individuals for diversity.
- **Dynamic Termination:** Stops the algorithm when the target sequence is achieved, ensuring efficiency.

Application

The binary string matching problem is a simplified example to demonstrate the working of GAs. This simulation can be extended to real-world applications, such as:

- **Digital System Optimization:**
  Optimizes binary configurations for hardware parameters like clock speeds, memory allocation, and performance tuning.

  In a digital system:

  - Bits at positions 1 and 2 control processing speed.
  - Bits at positions 3 and 4 determine memory usage.
  - The remaining bits adjust other critical performance parameters.

For a target sequence like "11010100", achieving an exact match signifies finding the optimal configuration for system performance. The genetic algorithm aids in exploring potential configurations systematically, adapting solutions through selection, crossover, and mutation.

Other applications could be

- **Error correction in data transmission:** Matching noisy binary data to its original form.
- **Genetic optimization in circuits:** Designing logic circuits by optimizing binary representations.

Conclusion

This project successfully demonstrated the working of a Genetic Algorithm through a simple yet effective example of binary string matching. The simulation highlighted the power of GAs in evolving solutions through biologically inspired operators.