BIO-INSPIRED COMPUTING (INFOBIO)

Chapter 02

Zouaidia Khouloud

CHAPTER 2: EVOLUTIONARY APPROACHES (30%)

1. Biological Foundations
2. Principles of Genetic
3. Algorithms Applications

INTRODUCTION TO EVOLUTIONARY APPROACHES

- Evolutionary approaches (or **evolutionary computation**) are a family of **artificial intelligence techniques** that use principles of **biological evolution** to solve complex computational problems, especially those that are too difficult or too large for traditional mathematical optimization methods.
- Instead of searching for an exact mathematical solution, evolutionary approaches **simulate the process of natural evolution**,where the best individuals survive, reproduce, and gradually evolve into better solutions over generations.
- In simple terms:
- Evolutionary algorithms “evolve” better answers over time, just as species evolve better adaptations in nature.

DARWIN'S CONCEPT OF NATURAL SELECTION

"In nature, individuals better adapted to their environment have a higher probability of survival and reproduction."

This principle is mapped into computational form:

- **Individuals** = Possible solutions to a problem.
- **Population** = A collection of candidate solutions.
- **Environment** = The problem or objective function defining fitness.
- **Fitness** = A measure of how good a solution is.
- **Reproduction** = Creating new candidate solutions from the best ones.
- **Mutation and Crossover** = Introducing variation, just like biological evolution.

Over many generations, the population *evolves*, producing increasingly optimal solutions.

HISTORICAL BACKGROUND

The Biological Inspiration : The roots lie in Charles Darwin's *On the Origin of Species* (1859), which introduced:

- *Natural Selection* : Favoring individuals with advantageous traits.
- *Variation* : Arising from mutation and recombination.
- *Inheritance* : Passing beneficial traits to offspring.

The Computational Development

In the 20th century, several researchers independently translated these biological ideas into computational algorithms:

HISTORY

Pioneer	Contribution	Year	Core Concept
Alan Turing	Suggested machines could “evolve” intelligence	1950	Early theoretical idea
John Holland	Developed the first formal *Genetic Algorithm (GA)*	1975	Reproduction, crossover, mutation
Ingo Rechenberg	Introduced *Evolution Strategies (ES)*	1965	Real-valued optimization
Lawrence Fogel	Created *Evolutionary Programming*	1966	Adaptive model prediction
John Koza	Proposed *Genetic Programming (GP)*	1992	Evolution of computer programs

COMPONENTS OF EVOLUTIONARY SYSTEMS

Every evolutionary approach, regardless of its variant, contains four essential components:

- **Population of Individuals:**
  A group of potential solutions, each represented by a *chromosome* (e.g., a string of numbers or symbols).
  Example: In optimizing a function f(x), each chromosome might represent a possible value of x.

- **Fitness Evaluation:**
  Determines how good each individual is at solving the problem.
  Example: For maximizing profit, fitness could be the total revenue generated by that solution.

- **Selection Mechanism:**
  Chooses which individuals will reproduce based on fitness (like "survival of the fittest").

- **Genetic Operators:**
  - **Crossover:** Combines information from two parents.
  - **Mutation:** Introduces small random changes for diversity.

This cycle: selection, reproduction, mutation, evaluation is repeated until the population stabilizes or a solution of acceptable quality is found.

EXAMPLE ANALOGY: EVOLUTION OF A BRIDGE DESIGN

- Imagine engineers trying to find the **most efficient bridge design**:
- Each bridge design = a chromosome (set of parameters: length, width, material).
- The **fitness** = how well it performs under load while minimizing cost.
- The GA starts with random designs.
- It selects the top performers, combines their traits (crossover), and randomly tweaks some (mutation).
- Over hundreds of generations, weaker designs are eliminated, and superior designs emerge.
- This mimics how **nature "searches" for the fittest organism**but here, it's the **fittest solution** to an engineering problem.

WHY USE EVOLUTIONARY APPROACHES?

Traditional optimization methods (like calculus-based techniques or gradient descent) often:

- Get stuck in local minima (non-global solutions)

Evolutionary algorithms, by contrast:

- Search **globally** and **stochastically**
- Handle **complex, non-smooth, and discontinuous** problems
- Are **domain-independent** so they can be applied to any problem that can be evaluated numerically

That's why they're widely used in **engineering, economics, machine learning, and even art.**

EXAMPLE – OPTIMIZING A MARKETING STRATEGY

Imagine a company wants to optimize its marketing campaign parameters (budget allocation, target demographics, channel choice).
There are **millions of possible combinations** , too many to test exhaustively.

A **Genetic Algorithm** can:

- Encode each possible strategy as a “chromosome”.
- Evaluate fitness based on past campaign performance metrics (ROI, conversions).
- Recombine and mutate parameters to find better combinations.
- Evolve highly effective strategies over iterations.

This approach saves computation and often finds **unexpected, creative solutions** that traditional models might miss.

BIOLOGICAL FOUNDATIONS

The field of **evolutionary computation** draws its inspiration directly from *biological evolution*, the natural process through which living organisms adapt and improve their chances of survival over successive generations.

- Organisms produce more offspring that can survive.
- Variations exist among individuals in a population.
- Those with advantageous traits are more likely to survive and reproduce.
- Over time, beneficial traits become more common.
- This process leads to **adaptation** , nature's way of solving complex optimization problems (e.g., finding efficient body structures, optimal behaviors, or survival strategies).
- Evolutionary algorithms replicate this **mechanism of adaptive improvement** , not with genes and DNA, but with data structures and fitness functions.

BIOLOGICAL CONCEPTS AND THEIR COMPUTATIONAL COUNTERPARTS

Biological Concept	Description	Computational Analogy in Evolutionary Algorithms
Chromosome / Genome	A string of DNA that encodes hereditary information	A data structure (string, vector, or tree) representing a potential solution
Gene	A unit of heredity controlling a trait	A parameter or component of a solution (e.g., a variable)
Allele	A possible value a gene can take	A value assigned to a variable or feature
Population	Group of individuals of a species	A set of candidate solutions
Fitness	Measure of reproductive success	Objective function value or performance measure
Mutation	Random change in DNA sequence	Random perturbation of a solution to introduce variation
Crossover (Recombination)	Mixing genes between parents	Combining parts of two solutions to form new ones
Natural Selection	Preferential survival of the fittest	Selecting better solutions for reproduction

The Genetic Code and Representation

In biology, the **genetic code** is composed of sequences of four nucleotides (A, T, G, C) that encode all the information for building an organism.

Similarly, in computational terms:

- Each solution is **encoded** as a "genetic string" (e.g., a binary sequence, array of numbers, or symbolic expression).
- This encoding determines the characteristics (phenotype) of the solution once it's "expressed" or evaluated.

Example

- **Biological case:**
  A gene sequence might determine eye color (blue, green, brown).
- **Computational analogy:**
  A string 1011 might represent a specific combination of variables in an optimization problem (e.g., temperature = high, pressure = medium, speed = low).

This encoding is crucial because the **search process happens in the genetic space**, not directly in the problem space.

Variation: The Source of Innovation

In biological evolution, *variation* ensures that populations do not stagnate and can adapt to changing environments.

There are two primary sources of genetic variation:

- **1. Mutation**

A random change in a gene during DNA replication (e.g., an A changes to a G).

- **Biological Role:** Introduces novelty; may produce advantageous traits.
- **Computational Role:** Randomly alters parts of a solution to explore new possibilities.

Example:

In a binary-coded GA, if a chromosome = 110010, a mutation might flip one bit → 110001.

- This prevents the algorithm from getting stuck in local optima.

Recombination (Crossover)

Combines genetic material from two parents to produce offspring.

- **Biological Role:** Produces diversity and allows beneficial traits to mix.
- **Computational Role:** Combines partial solutions to produce better offspring.

Example:

- Parent 1: 110011
- Parent 2: 101100
- Crossover point: between 3rd and 4th bit
- Offspring: 110100

This simulates *genetic inheritance*, where the child inherits some traits from both parents.

Natural Selection and Fitness

In nature, only individuals that are better adapted to their environment tend to survive and reproduce , this is **natural selection**.

- In evolutionary algorithms, this is formalized as the **fitness function**, which evaluates how good a solution is relative to others.

Example:

If you're optimizing a function $f(x) = x^2$:

- A chromosome representing $x = 2$ has a fitness of 4.
- A chromosome representing $x = 5$ has a fitness of 25.
- The latter is "fitter" and more likely to be selected for reproduction.

- **Selection strategies** mimic different evolutionary pressures:
- **Roulette Wheel Selection:** Probability of selection $\propto$ fitness.
- **Tournament Selection:** Randomly compete individuals; the better one wins.
- **Elitism:** The best individuals automatically carry over to the next generation.

This creates a **competitive yet adaptive environment**, ensuring steady progress toward better solutions.

Survival, Adaptation, and Evolutionary Dynamics

In both nature and computation, populations evolve through a balance between:

- **Exploitation:** Using the best solutions found so far.
- **Exploration:** Trying new, untested combinations.
- This mirrors the **biological balance** between stability and change:
- Too much mutation → chaos, no convergence.
- Too little mutation → stagnation, premature convergence.

Over generations, the population evolves **toward higher fitness**, analogous to how species evolve to become better suited to their environments.

EXAMPLE ANALOGY: EVOLUTION OF BIRDS' BEAKS

- Birds with slightly different beak shapes had advantages depending on available food.
- Over generations, environmental pressure caused finches to evolve distinct beak types optimized for specific diets.
- **Computational analogy:**
- Each "bird" = a candidate solution.
- "Beak shape" = a parameter (decision variable).
- "Environment" = the optimization problem.
- Over time, the population evolves beak shapes (solutions) that best fit the problem (environmental constraints).
- This biological adaptation directly parallels how evolutionary algorithms refine solutions through fitness-driven selection and variation.

THE ROLE OF INHERITANCE

- In both biology and computation, **inheritance** ensures that beneficial traits (or solution features) are preserved and passed to future generations.
- In genetic algorithms:
- Information from good solutions is reused (via crossover and elitism).
- This **accumulates knowledge**, making the population progressively better.
- This inheritance mechanism prevents the loss of good traits, enabling **long-term optimization**.

EVOLUTION AS A SEARCH AND OPTIMIZATION PROCESS

Nature can be viewed as a **massively parallel optimization system**:

- Millions of species = diverse solutions.
- Each generation refines adaptations incrementally.
- The fitness landscape = environment (constraints + opportunities).
- Evolution = iterative improvement toward local or global optima.
- Similarly, **evolutionary computation** searches through a fitness landscape:
- Each point represents a potential solution.
- The goal is to find global peaks (best solutions).
- This analogy explains why evolutionary algorithms are effective in problems where **the search space is vast, complex, or poorly understood.**

Biological Process	Evolutionary Computation Equivalent	Purpose
Natural selection	Fitness-based selection	Keeps best solutions
Genetic mutation	Random solution alteration	Introduces diversity
Reproduction	Crossover / recombination	Mixes good traits
Heredity	Encoding and inheritance of traits	Preserves success
Adaptation	Convergence toward optimal solutions	Problem-solving
Evolution	Iterative improvement of population	Optimization process

Principles of Genetic Algorithms

- A **Genetic Algorithm (GA)** is a **search and optimization technique** inspired by the process of natural evolution.

- It was formally introduced by **John Holland (1975)** at the University of Michigan, based on Darwinian principles of *natural selection* and *genetics*.
- In essence, GAs **simulate evolution** in a computer program to discover good or optimal solutions to problems where:

- The search space is large and complex.
- There are many possible solutions.
- There is no straightforward way to compute the best answer.
- "A Genetic Algorithm searches for optimal solutions by evolving a population of individuals using selection, crossover, and mutation , just like nature evolves species."

The GA Cycle (The Evolutionary Loop)

A GA evolves a population of solutions over generations.

- The general process is:
- **Initialization**
  Generate a random initial population of potential solutions (chromosomes).
- **Evaluation**
  Each chromosome is *decoded* and *evaluated* using the fitness function, a measure of how good the solution is.
- **Selection**
  Choose fitter individuals to be parents based on their fitness scores.
- **Crossover (Recombination)**
  Combine parent solutions to create offspring, sharing features from both.
- **Mutation**
  Introduce small random changes to some offspring to maintain diversity.
- **Replacement**
  Replace less fit individuals in the population with new offspring.
- **Termination**
  The cycle repeats until a stopping condition is met (e.g., maximum generations, desired fitness achieved, or convergence).
- **Illustrative Example**

ILLUSTRATIVE EXAMPLE

Let's use a simple optimization problem:

> Maximize $f(x) = x^2$, where $x \in [0, 31]$.

Step 1: Encoding

Represent x as a 5-bit binary string.

Example:

`x = 13` → Binary: `01101`

Step 2: Initial Population

Generate random chromosomes:

```
makefile

P1: 01010 (x=10, f=100)
P2: 10100 (x=20, f=400)
P3: 01101 (x=13, f=169)
P4: 11111 (x=31, f=961)
```

Step 3: Selection

Fitter individuals (higher f(x)) have higher chances of being selected.

`P4` and `P2` will likely be chosen as parents.

Step 4: Crossover

Single-point crossover between `P4 (11111)` and `P2 (10100)` at position 3:

```yaml
Parent 1: 111|11
Parent 2: 101|00
Offspring 1: 11100 (x=28)
Offspring 2: 10111 (x=23)
```

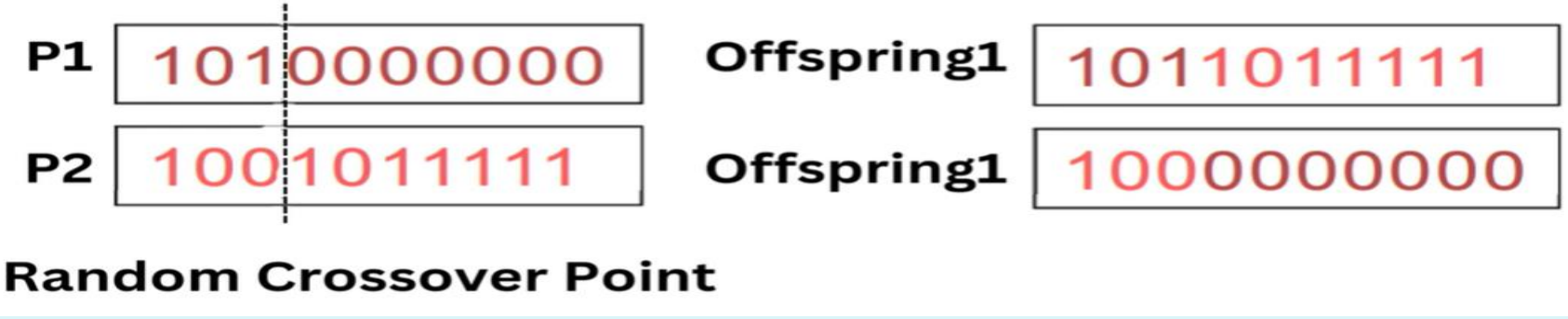
Step 5: Mutation

Flip one bit in Offspring 2:

```scss
10111 → 11111 (x=31)
```

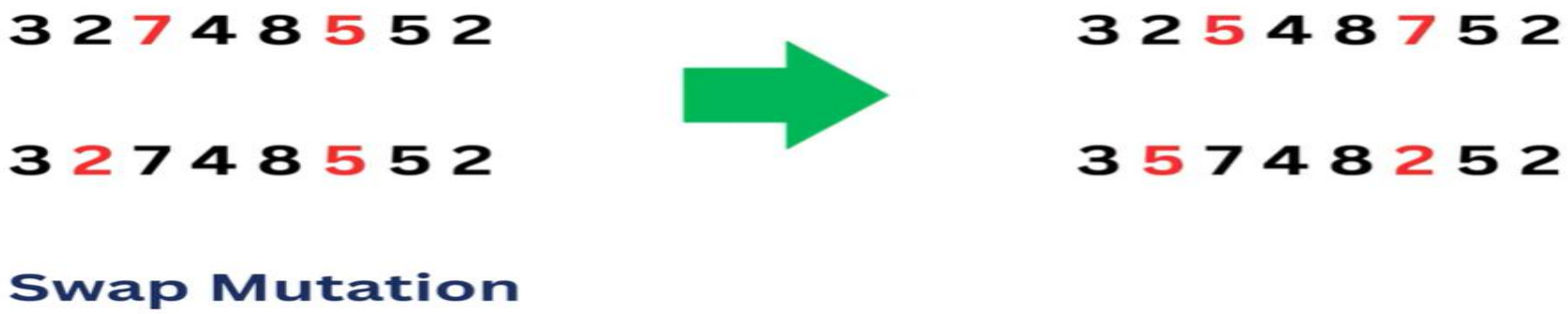
Step 6: Replacement

New generation might replace the worst individuals. Over several generations, the population toward the maximum $x = 31$, $f(x) = 961$.



Random Crossover Point

Two Point Crossover



Swap Mutation

THE FITNESS FUNCTION

- The **fitness function** is the driving force of a GA.
  It determines which individuals are more likely to survive and reproduce.
- Mathematically, it's a mapping: $Fitness : \text{Solution} \rightarrow \text{Real number}$
- A good fitness function should:
- Reflect the goal of optimization clearly.
- Provide distinguishable values for better/worse solutions.
- Be computationally efficient to evaluate.

Example:
For minimizing the total cost : $Fitness(x) = \frac{1}{1 + C(x)}$

- so that lower cost → higher fitness.

SELECTION MECHANISMS

- Selection determines **who gets to reproduce.**
  It biases reproduction toward fitter individuals but maintains some randomness to preserve diversity.
- **Example:** In *Roulette Selection*, if total fitness = 200 and an individual's fitness = 50, it has a 25% chance of selection.

Selection Type	Mechanism	Characteristics
Roulette Wheel Selection	Probability proportional to fitness	Simple, but can favor elites too strongly
Tournament Selection	Randomly select subset, choose fittest	Good balance of exploration/exploitation
Rank Selection	Based on rank order, not raw fitness	Prevents domination by outliers
Elitism	Automatically carry over best solutions	Guarantees best individuals survive
Stochastic Universal Sampling	Uses multiple selection points for fairness	Reduces selection bias

MUTATION

- Mutation introduces small random changes to offspring to ensure diversity and prevent premature convergence.

Mutation Type	Example	Purpose
Bit Flip Mutation	11001 → 11101	Binary encoding
Swap Mutation	[1, 2, 3, 4] → [1, 3, 2, 4]	Permutation encoding
Gaussian Mutation	Add random noise	Real-valued encoding
Scramble Mutation	Randomly reorder subset	Combinatorial optimization

REPLACEMENT AND GENERATIONAL MODELS

- After new offspring are produced, the GA decides how to form the next generation.
- **Generational Replacement:** Replace the entire population.
- **Steady-State Replacement:** Replace only a few weakest individuals.
- **Elitism:** Retain best individuals to prevent loss of good solutions.
- These strategies balance *innovation (exploration)* and *preservation (exploitation).*

CONVERGENCE AND STOPPING CRITERIA

A GA typically stops when:

- A maximum number of generations is reached.
- No improvement in fitness is observed over time.
- A desired fitness level is achieved.
- Population diversity drops below a threshold.
- Convergence indicates that the algorithm has found a region of high-quality solutions ,possibly near a global optimum.

PRACTICAL EXAMPLE: THE TRAVELING SALESMAN PROBLEM (TSP)

- **Problem:**
  Find the shortest route visiting all cities once and returning to the start.
- **GA Setup:**
- **Encoding:** Each chromosome = sequence of cities.
  Example: [A, D, B, C]
- **Fitness Function:**

$$F = 1/(\text{Total distance})$$

- **Crossover:** Partially mapped crossover (PMX) preserves city order.
- **Mutation:** Swap two cities.
- **Selection:** Tournament selection.
- **Result:** After 100 generations, GA converges to near-optimal route.
- This approach outperforms brute-force search when there are hundreds of cities.

ADVANCED VARIANTS

- **Adaptive GAs:** Adjust crossover and mutation rates dynamically.
- **Hybrid GAs:** Combine with local search (e.g., hill climbing).
- **Parallel GAs:** Distribute populations over multiple processors.
- **Multi-objective GAs (MOGA):** Handle problems with several conflicting goals (e.g., NSGA-II algorithm).
- These extensions allow GAs to tackle modern challenges in engineering, AI, and complex system design

APPLICATIONS OF EVOLUTIONARY ALGORITHMS

- **Engineering Optimization**

Used to optimize structures, aerodynamics, and energy systems.
Example: NASA used GAs to evolve satellite antenna designs.

- **Machine Learning** , GAs optimize:

1. Neural network weights
2. Feature selection
3. Hyperparameters

- **Bioinformatics** Used for:

1. Gene sequence alignment
2. Protein folding prediction
3. Drug design,

- **Economics and Game Theory**

Evolutionary strategies model agent behavior and adaptive markets.

- **Robotics and Control Systems**

Evolutionary algoevolve control parameters or behaviors for autonomous agents

- **Creative Arts** Used in:

Generative design for music,architecture,and digital art by evolving aesthetic patterns.

FUTURE DIRECTIONS

- **Hybrid Evolutionary Methods:** Combining GAs with neural networks or reinforcement learning.
- **Quantum-Inspired Algorithms:** Exploiting quantum principles for faster convergence.
- **Coevolutionary Systems:** Simulating ecosystems of competing and cooperating agents.
- **AutoML & AI Governance:** GAs for automated model selection and ethical decision optimization.