Tutorial on Genetic Algorithms (GA)

Advanced Algorithms

1. Introduction

A **Genetic Algorithm (GA)** is a population-based metaheuristic inspired by the process of natural selection and genetics. It is widely used for complex optimization problems where traditional gradient-based techniques fail or the search space is discontinuous.

The GA evolves a set of possible solutions (*chromosomes*) through processes analogous to biological evolution: **selection**, **crossover**, and **mutation**.

$$\text{Population} \rightarrow \text{Selection} \rightarrow \text{Crossover} \rightarrow \text{Mutation} \rightarrow \text{Next Generation}$$

—

2. Basic Components of a GA

2.1 Representation (Encoding)

Each solution is encoded as a **chromosome**:

- **Binary Encoding:** Solution represented as a bit string, e.g. $S = [1, 0, 1, 1, 0]$.
- **Permutation Encoding:** For ordering problems (e.g., TSP), $S = [A, B, C, D, E]$.
- **Real-valued Encoding:** Used for continuous problems, $S = [2.4, 1.6, 3.7]$.

—

3. Phases of a Genetic Algorithm

3.1 Initialization

The algorithm starts with a randomly generated population of N individuals (solutions).

$$P(0) = \{S_1, S_2, \ldots, S_N\}$$

Each chromosome is typically generated randomly within permissible ranges.

Example: If we are minimizing $f(x) = x^2$ for $x \in [0, 31]$, we can encode x using 5-bit binary strings:

$$x = [00000], [00101], [11111], [01010], \ldots$$

—

3.2 Fitness Evaluation

Each chromosome is evaluated using a **fitness function** that measures how good it is.

If we are minimizing a cost function $f(S)$, then fitness can be defined as:

$$F(S) = \frac{1}{1 + f(S)}$$

so that lower cost corresponds to higher fitness.

Example: For $f(x) = x^2$ and population $\{1, 2, 3\}$:

$$f(1) = 1, \quad f(2) = 4, \quad f(3) = 9$$

$$F(1) = \frac{1}{2},\ F(2) = \frac{1}{5},\ F(3) = \frac{1}{10}$$

—

3.3 Selection

Selection chooses parent chromosomes for reproduction based on fitness.

Goal: Better solutions have a higher chance to pass their genes to the next generation.

Common Selection Methods:

- **Roulette Wheel Selection:** Probability of selection proportional to fitness.

  $$P(S_i) = \frac{F(S_i)}{\sum_j F(S_j)}$$

  Example: If total fitness = 0.8, and $F(S_1) = 0.4$, then $P(S_1) = 0.5$.

- **Tournament Selection:** Randomly pick k individuals and select the best among them.

  Example: For $k = 3$, choose random 3 chromosomes $\{S_2, S_5, S_7\}$, and select the one with highest fitness.

- **Rank Selection:** Sort population by fitness, assign selection probability based on rank (not raw fitness).

Purpose: Selection implements the "*survival of the fittest*" principle.

—

3.4 Crossover (Recombination)

Crossover combines parts of two parent chromosomes to create offspring. It is the main mechanism for **exploitation** — recombining good traits.

Common Types:

- **Single-Point Crossover:** Select a random crossover point and exchange tails.

$$\begin{array}{c} \text{Parent 1: } [1\ 0\ 1 \mid 0\ 0\ 1] \\ \text{Parent 2: } [0\ 1\ 0 \mid 1\ 1\ 0] \\ \hline \text{Child 1: } [1\ 0\ 1\ 1\ 1\ 0], \quad \text{Child 2: } [0\ 1\ 0\ 0\ 0\ 1] \end{array}$$

- **Two-Point Crossover:** Two crossover points; exchange middle segment.
- **Uniform Crossover:** Each gene is independently chosen from one of the parents with equal probability.
- **Order Crossover (OX):** Used for TSP and permutation problems. Two cut points define a segment copied from one parent, and remaining elements are filled in order from the second parent.

Typical crossover probability:

$$P_c = 0.8 \text{ to } 0.9$$

—

3.5 Mutation

Mutation introduces random changes to maintain diversity and avoid local optima. It is the main mechanism for **exploration**.

Common Types:

- **Bit-Flip Mutation (Binary Encoding):** Randomly flips a bit.

$$[1\ 0\ 1\ 0] \rightarrow [1\ 1\ 1\ 0]$$

- **Swap Mutation (Permutation):** Swaps two positions in a tour or sequence.

$$[A\ B\ C\ D\ E] \rightarrow [A\ D\ C\ B\ E]$$

- **Inversion Mutation:** Reverses a selected segment.

$$[A\ B\ |C\ D\ E|F] \rightarrow [A\ B\ E\ D\ C\ F]$$

- **Gaussian Mutation (Real-Valued):** Adds small Gaussian noise:

$$x_i' = x_i + \mathcal{N}(0, \sigma^2)$$

Typical mutation probability:

$$P_m = 0.01 \text{ to } 0.05$$

—

3.6 Replacement (Forming Next Generation)

After crossover and mutation, we form a new generation $P(t+1)$.

Common strategies:

- **Generational Replacement:** Replace entire population.
- **Elitism:** Keep a few best individuals from previous generation.

Elitism Example: If the best fitness in generation t is 0.95, that individual is directly copied to $P(t+1)$ to ensure best solution is never lost.

—

4. Example: Binary GA for $f(x) = x^2$

Chromosome	x	$f(x) = x^2$
00000	0	0
00101	5	25
01010	10	100
01111	15	225

$$F(x) = \frac{f(x)}{\sum f(x)} \Rightarrow \text{Higher } f(x) \text{ means better selection chance.}$$

Crossover between 01010 and 01111 at position 3:

$$[010|10] \ + \ [011|11] \Rightarrow [01011], [01110]$$

Mutation (bit-flip) on last bit of 01011:

$$01011 \rightarrow 01010$$

After several generations, population converges to $x = 31$, $f(x) = 961$.

—

5. Advantages and Key Features

- Works with both continuous and discrete search spaces.
- Does not require gradient information.
- Global search capability through population-based exploration.
- Crossover promotes exploitation, mutation ensures exploration.
- Can be hybridized with other heuristics for faster convergence.

—

6. Typical Flow of a Genetic Algorithm

Step 1: Initialize population randomly.
Step 2: Evaluate fitness of each individual.
Step 3: Select parents based on fitness.
Step 4: Apply crossover with probability P_c.
Step 5: Apply mutation with probability P_m.
Step 6: Form new population using elitism or replacement.
Step 7: If termination condition met, stop; else repeat.

—

7. Concluding Remarks and Research Directions

- **Adaptive GA:** Dynamically changes P_c and P_m during search.
- **Hybrid GA:** Combines GA with local search or simulated annealing.
- **Parallel GA:** Multiple sub-populations evolve in parallel and exchange individuals.
- **Applications:** Scheduling, routing (TSP), neural network training, feature selection, parameter tuning.

—