Genetic Algorithms

Shailesh Sharma

July 3, 2025

What is a Genetic Algorithms?

Genetic Algorithms (GA) are search algorithms based on the mechanics of natural selection. It is used to solve complex problems by searching through large and complex spaces for optimal or near-optimal solutions.

Problems Faced Before Genetic Algorithms (GAs)

Before GAs, solving complex optimization problems was difficult due to:

- **Slow or Inefficient Methods**: Traditional techniques (e.g., brute-force) were slow or got stuck in local optima.
- **Limited Applicability**: Many methods required convex, smooth, or differentiable functions.
- **Problem-Specific Approaches**: Algorithms were often tailored to specific problems and lacked generality.
- **Poor Scalability**: Classical methods struggled with large, combinatorial search spaces.

How GAs Helped

Genetic Algorithms provided:

- A general-purpose optimization method.
- Ability to handle multimodal, non-differentiable, and noisy problems.
- Population-based search to avoid local optima.

Example

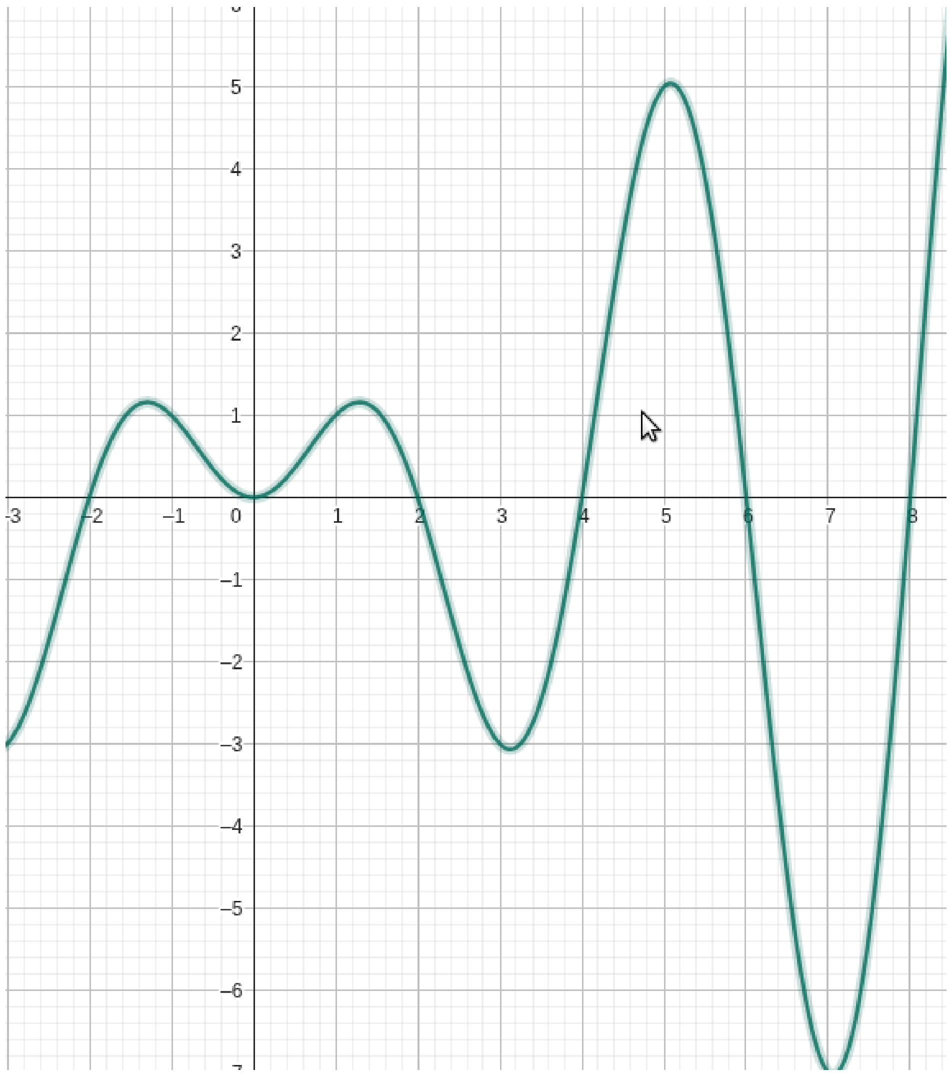
How to minimize the $f(x) = x \cdot \sin(90x)$

Objective

Maximize the function:

$$f(x) = x \cdot \sin(\frac{\pi}{2}x) \quad \text{for } x \in [-2, 8]$$

Graph



Function Characteristics

- **Oscillatory**: Due to the sine term.
- Many local maxima and minima.
- Nonlinear and Non-convex.

Convex and Non-Convex in Genetic Algorithm

The concepts of convex and non-convex play an important role in shaping the difficulty and behavior of the solution search.

Convex Problems

- Traditional methods like Gradient Descent or Linear Programming work well.

- Genetic Algorithm is not strictly needed, but still works.
- One global optimum.

Non-Convex Problems

- Much harder for classical methods (get stuck in local minima).
- Their population-based and probabilistic nature allows them to avoid getting stuck in local minima.

Genetic Algorithm Steps

Step 1: Initialization
Generate an initial population of individuals randomly. Each individual represents a possible solution.

Step 2: Evaluation
Calculate the fitness of each individual using a fitness (objective) function, which reflects the quality of the solution. Ex. $\text{fitness}(x) = f(x) = x \cdot \sin(\frac{\pi}{2}x)$

Step 3: Selection
Select individuals from the current population based on their fitness to become parents. Higher fitness increases the probability of selection. Common methods include:

- Roulette Wheel Selection
- Tournament Selection
- Rank Selection

Step 4: Crossover (Recombination)
Combine the genetic information of two parents to generate offspring. Types of crossover:

- Single-point crossover
- Two-point crossover
- Uniform crossover
- Arithmetic crossover (for real-valued genes)

Step 5: Mutation
Apply random changes to offspring to maintain diversity. Types include:

- Bit-flip mutation (binary)
- Gaussian or uniform perturbation (real-valued)

Step 6: Replacement
Replace individuals in the population with the new offspring. Strategies:

- Generational replacement (replace all)
- Steady-state replacement (replace a few)
- Elitism (preserve the best individuals)

Step 7: Termination

Repeat Steps 2 to 6 until a stopping criterion is met:

- Maximum number of generations reached
- Fitness plateau (no significant improvement)
- Satisfactory solution found

Now We will setup Problem to understand the GA's working

- **Objective:** Maximize
$$f(x) = x \cdot \sin\left(\frac{\pi}{2}x\right)$$
- **Domain:** $x \in [0, 8]$
- **Chromosome encoding:** 8-bit binary string
Map binary string b to real x by:
$$x = \frac{\text{int}(b)}{255} \times 8$$
- **Population size:** 4
- **Generations:** 3
- **Fitness:** Here, fitness = f(x) (since we maximize).
- **Crossover:** Single-point
- **Mutation:** Flip 1 random bit per child per generation

Generation 0 — Initialization

ID	Chromosome	Decimal	x	$f(x) = x\sin\left(\frac{\pi}{2}x\right)$	Fitness
A	01010101	85	2.667	$2.667 \times \sin(4.188) \approx -2.103$	-2.103
B	11110000	240	7.529	$7.529 \times \sin(11.825) \approx -5.148$	-5.148
C	00110011	51	1.600	$1.600 \times \sin(2.513) \approx 0.94$	0.94
D	10000001	129	4.047	$4.047 \times \sin(6.357) \approx 0.382$	0.382

Note: Since some fitness values are negative, we shift all fitnesses by an offset to keep them positive for selection:

$$\text{offset} = |\min f(x)| + \epsilon = 5.148 + 1 = 6.148$$

Adjusted fitness values:

Step 1: Selection (Tournament or Roulette Wheel)

- Select parents with higher adjusted fitness values.
- Example parents: C (9.628) and D (7.666)

ID	$f(x)$	Adjusted fitness $= f(x) + 6.148$
A	-2.103	4.045
B	-5.148	1.000
C	0.94	7.088
D	0.382	6.53

Step 2: Crossover (Single-point at bit 4)

Parent C:	0011 0011
Parent D:	1000 0001
Offspring 1:	0011 0001
Offspring 2:	1000 0011

Step 3: Mutation (flip 1 random bit)

- O1: Flip bit 2 $\Rightarrow$ 0001 0001
- O2: Flip bit 7 $\Rightarrow$ 1000 0010

Step 4: Evaluate New Population

ID	Chromosome	Decimal	x	$f(x)$	Adjusted fitness
O1	00010001	17	0.533	$0.533 \times \sin(0.837) \approx 0.392$	6.54
O2	10000010	130	4.078	$4.078 \times \sin(6.405) \approx 0.447$	6.595
A	01010101	85	2.667	-2.103	4.045
B	11110000	240	7.529	-5.148	1.000

Generation 1 Selection

Best individuals for next selection: O1 (6.54), O2 (6.595)
Parents: O1 and O2

Crossover (single-point at bit 5)

O1:	00010 001
O2:	10000 010
Offspring 3:	00010 010
Offspring 4:	10000 001

Mutation (flip 1 bit)

- O3: Flip bit 0 $\Rightarrow$ 10010 010
- O4: Flip bit 4 $\Rightarrow$ 10001 001

Evaluate New Population

ID	Chromosome	Decimal	x	$f(x)$	Adjusted fitness
O3	10010010	146	4.576	$4.576 \times \sin(7.189) \approx 3.182$	9.33
O4	10001001	137	4.294	$4.294 \times \sin(6.742) \approx 1.626$	7.774
A	01010101	85	2.667	-2.103	4.045
B	11110000	240	7.529	-5.148	1.000

Best for the next genetation: O3 (7.245), O4 (6.646)

- Fitness values were shifted to keep positive values for selection.
- The best found solution here is approximately $x = 4.294$ with $f(x) \approx 0.866$.
- With a small population and few generations, the GA explored limited regions.

1. What is a Schema?

So a GA isn't just searching individual solutions — it's searching entire families of solutions described by schemas.

A **schema** H is a template that represents a subset of binary strings. It uses symbols from $\{0, 1, *\}$, where:

- `0`, `1`: fixed positions
- `*`: wildcard that can be either 0 or 1

Example: Let

$$H = 1 * 0 * **$$

This schema matches all binary strings of length 6 that:

- Have a `1` at position 1,
- A `0` at position 3,
- And any values in all other positions.

2. Schema Properties

- **Order:** $o(H)$ = number of fixed positions (non-`*`).
- **Length:** $\delta(H)$ = distance between the first and last fixed positions.
- **Fitness:** $f(H)$ = average fitness of strings matching schema H.

Example: For $H = 1 * 0 * **$:

$$o(H) = 2, \quad \delta(H) = 3 - 1 = 2$$

3. Schema Theorem

Let:

- $m(H, t)$: number of individuals matching schema H at generation t,
- $\bar{f}$: average fitness of population,
- $f(H)$: average fitness of individuals matching H,
- p_c: crossover probability,
- p_m: mutation probability (per bit),
- l: length of chromosome.

Then the expected number of individuals matching schema H in generation $t + 1$ is bounded by:

$$E[m(H, t+1)] \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \cdot \left[1 - p_c \cdot \frac{\delta(H)}{l-1} - o(H) \cdot p_m\right]$$

4. Interpretation of the Theorem

- Schema with above-average fitness $\left(\frac{f(H)}{\bar{f}} > 1\right)$ are likely to increase in population.
- Short ($\delta(H)$ small) and low-order ($o(H)$ small) schema are less disrupted by crossover and mutation.

Genetic Algorithm Operators

In Genetic Algorithms (GA), **operators** are mechanisms that modify individuals (solutions) to guide the search toward optimal solutions. The main operators in GA are:

1. Selection Operator

- **Purpose:** Selects individuals (parents) for reproduction based on fitness.
- **Examples:** Roulette wheel selection, tournament selection, rank selection, stochastic universal sampling.

Interpretation: Determines *who gets to be the parents.*

2. Crossover Operator (Recombination)

- **Purpose:** Combines two parents to create offspring.
- **Examples:** Single-point crossover, two-point crossover, uniform crossover, arithmetic crossover (for real-valued genes).

Interpretation: Determines *how two solutions are mixed to form new ones.*

3. Mutation Operator

- **Purpose:** Introduces small random changes to maintain genetic diversity.
- **Examples:** Bit flip mutation (binary), Gaussian mutation (real-valued), swap mutation (permutation problems).

Interpretation: Determines *how to explore new areas of the search space.*

4. Replacement Operator

- **Purpose:** Decides how to form the next generation population.
- **Examples:** Generational replacement, elitism, steady-state replacement.

Formal Proof of Nonconvexity and Discontinuity in Ride-Sharing Optimization

Problem Setup

Let:

- $D = \{d_1, d_2, \ldots, d_{|D|}\}$ be the set of drivers,
- $R = \{r_1, r_2, \ldots, r_{|R|}\}$ be the set of riders.

Define the binary decision variable:

$$I_{d_i,r_j} \in \{0,1\}, \quad \forall d_i \in D,\ \forall r_j \in R,$$

with the interpretation:

- $I_{d_i,r_j} = 1$: rider r_j is assigned to driver d_i,
- $I_{d_i,r_j} = 0$: otherwise.

Let the full vector of decision variables be:

$$\mathbf{I} = \left(I_{d_1,r_1}, I_{d_1,r_2}, \ldots, I_{d_{|D|},r_{|R|}}\right) \in \{0,1\}^n, \quad \text{where } n = |D| \cdot |R|.$$

The objective is to maximize a linear function:

$$\max \sum_{i=1}^{|D|} \sum_{j=1}^{|R|} I_{d_i,r_j}.$$

Subject to Constraints

1. **Unique Assignment Constraint:**

$$\sum_{i \in D} I_{d_i,r_j} \leq 1, \quad \forall j \in \{1, 2, \ldots, |R|\} \tag{1}$$

2. **Driver Capacity Constraint:**

$$\sum_{j \in R} I_{d_i,r_j} \leq n_i, \quad \forall i \in \{1, 2, \ldots, |D|\} \tag{2}$$

3. **Path Deviation Constraint:**

$$\Delta_i \leq t_i \cdot |P_i|, \quad \forall i \in \{1, 2, \ldots, |D|\} \tag{3}$$

Convex Optimization Problem (Standard Form)

This definition is from *Convex Optimization*, pages 136–137.

Definition:

A convex optimization problem is one of the form:

$$\begin{aligned} \text{Minimize} \quad & f_0(x) \\ \text{Subject to} \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ & a_i^T x = b_i, \quad i = 1, \ldots, p, \end{aligned}$$

where:

- $f_0(x)$ is the **objective function** (to be minimized),
- $f_i(x)$ are **inequality constraint functions**,
- $a_i^T x = b_i$ are **equality constraints**.

Convexity Requirements

To qualify as a convex optimization problem, the following must hold:

1. $f_0(x)$ is a **convex function** over a **convex feasible set**,
2. Each inequality constraint function $f_i(x)$ is **convex**,
3. Each equality constraint $a_i^T x = b_i$ is **affine** (i.e., linear).

Convex function: $f(x_1, x_2, \ldots, x_n)$ is convex if, for each pair of points on the graph of f, the line segment joining these two points lies entirely above or on the graph of f.

Convex set: A convex set is a collection of points such that, for each pair of points in the collection, the entire line segment joining these two points is also in the collection.

Both definitions are from *Introduction to Operations Research* by Frederick S. Hillier and Gerald J. Lieberman, pages 995 and 997.

2. Proof of Nonconvexity (by Counterexample)

We demonstrate nonconvexity by constructing a specific counterexample.

2.1 Simplified Problem

Consider a minimal problem with:

- One driver d_1, i.e., $|D| = 1$,
- Two riders r_1 and r_2, i.e., $|R| = 2$,
- $x \equiv I_{d_i, r_j}$ and objective

$$f_0(x) = f_0(I_{d_i, r_j}) = -\sum_{i=1}^{|D|} \sum_{j=1}^{|R|} I_{d_i, r_j},$$

- All constraints are trivially satisfied (e.g., unlimited capacity).

The decision variables reduce to:

$$I_{d_1,r_1}, \quad I_{d_1,r_2} \in \{0,1\}.$$

Convexity Requirement

Condition 1: $f_0(I_{d_i,r_j})$ is a **convex function** over a **convex feasible set**.

First, we prove that the feasible set is convex, and then we prove $f_0(I_{d_i,r_j})$ is a convex function. So the feasible region is:

$$\mathcal{C} = \{(0,0),(1,0),(0,1),(1,1)\} \subseteq \{0,1\}^2.$$

According to the definition of a convex set, let:

$$x = (1,0) \in \mathcal{C}, \quad y = (0,1) \in \mathcal{C}.$$

Take a convex combination with $\theta = 0.5$. When $\theta = 0$, you get y; when $\theta = 1$, you get x; and when $0 < \theta < 1$, you get points strictly between x and y:

$$z = \theta x + (1-\theta)y = 0.5 \cdot (1,0) + 0.5 \cdot (0,1) = (0.5, 0.5).$$

Contradiction

Clearly,

$$z = (0.5, 0.5) \notin \mathcal{C} = \{0,1\}^2.$$

Therefore, the set $\mathcal{C}$ is **not** a convex set. Hence, Ride-Sharing Optimization is **not a convex problem**. We will not check for other requirements.

Note:

1. The feasible sets for discrete optimization problems can be thought of as exhibiting an extreme form of nonconvexity, as a convex combination of two feasible points is in general not feasible.(Nocedal and Wright, *Numerical Optimization*, 2nd ed., Springer, 1999, p. 5).

2. Both ILP and BILP are fundamental discrete optimization models because many real-world problems can be modeled using integer or binary decisions.

Maximizing Rider Accommodation While Ensuring Fairness Among Drivers Using NSGA-II

Background

We address the problem of optimally assigning riders to drivers with two conflicting objectives:

1. Maximizing the total number of riders accommodated.
2. Minimizing the variance in rider assignments across drivers to promote equitable workload distribution.

Problem Statement

Let:

- $D = \{d_1, \ldots, d_m\}$: Set of available drivers.
- $R = \{r_1, \ldots, r_n\}$: Set of riders requesting service.

Each driver $d_i \in D$ is characterized by:

- Origin and destination nodes: s_i, f_i.
- Shortest path length: $|P_i|$.
- Vehicle capacity: n_i.
- Permissible deviation threshold: $t_i \in [0, 1]$.

Each rider $r_j \in R$ can be assigned to at most one compatible driver, subject to feasibility based on route deviation and driver capacity.

Decision Variables

- $x_{ij} \in \{0, 1\}$: Binary variable, equals 1 if rider r_j is assigned to driver d_i, and 0 otherwise.
- Δ_i: Total additional distance traveled by driver d_i to accommodate assigned riders.

Objectives

- $f_1(x) = \sum_{i \in D} \sum_{j \in R} x_{ij}$
- $f_2(x) = \frac{1}{|D|} \sum_{i \in D} \left(\sum_{j \in R} x_{ij} - \frac{1}{|D|} \sum_{i \in D} \sum_{j \in R} x_{ij} \right)^2$

Goal

- **Objective 1:** Maximize Rider Accommodation

$$\min \left(-f_1(x) \right) \tag{4}$$

- **Objective 2:** Minimize Variance for Fairness

$$\min f_2(x) \tag{5}$$

Constraints

C1. Unique Assignment Constraint (each rider assigned to at most one driver):

$$\sum_{i \in D} x_{ij} \leq 1 \quad \forall j \in \{1, \ldots, |R|\} \tag{6}$$

C2. Capacity Constraint (driver cannot exceed vehicle capacity):

$$\sum_{j \in R} x_{ij} \leq n_i \quad \forall i \in \{1, \ldots, |D|\} \tag{7}$$

C3. Route Deviation Constraint (limit on additional travel for drivers):

$$\Delta_i \leq t_i \cdot |P_i| \quad \forall i \in \{1, \ldots, |D|\} \tag{8}$$

Note: Δ_i denotes the extra distance incurred by driver d_i due to deviations from the original route to accommodate assigned riders.

Assumptions

To ensure a well-defined and tractable optimization model, we adopt the following assumptions:

1. **No Partial Routes or Transfers:** Each rider must be fully served by a single driver; no intermediate transfers or split routes are allowed.

2. **No Shared Route Benefit Between Riders:** The addition of one rider does not reduce the marginal cost of serving another; each rider's impact on deviation and capacity is evaluated independently.

3. **No Post-Drop-off Pickup:** Once a driver completes a trip for a set of assigned riders, they do not pick up additional riders mid-journey or after drop-offs.

4. **Homogeneous Capacity Assumption:** All drivers have equal or nearly equal vehicle capacities, and each driver's capacity is sufficient to accommodate the average number of riders ($n_i \geq \mu$). This ensures that fairness can be measured directly via variance in the number of assigned riders without capacity normalization.

5. **Unique Assignment:** Each rider is assigned to at most one compatible driver, based on feasibility determined by the route deviation and the driver's remaining capacity.

Objective Function: Min–Max Scalarization

Definitions

Used in multi-objective optimization to convert conflicting objectives into a single scalar objective by minimizing the worst (maximum) normalized objective value:

$$\min_x \max_i f_i^{\text{norm}}(x)$$

Used in robust optimization, game theory, or adversarial optimization, where you optimize against the worst-case scenario.

$$\max_x \min_y \phi(x, y)$$

There are saveral types of scalarization techniques used in multi-objective optimization to reduce multiple conflicting objectives into a single objective. These definitions are from *Adaptive Scalarization Methods in Multiobjective Optimization* by Gabriele Eichfelder.

1. **Weighted Sum Scalarization** Combine objectives linearly.

$$F(x) = \sum w_i f_i(x)$$

   Pros: Simple, fast.
   Cons: Misses non-convex parts of the Pareto front.

2. **ϵ -Constraint Scalarization** Optimize one objective, turn others into constraints.

$$\min f_1(x) \quad \text{subject to} \quad f_i(x) \leq \epsilon_i$$

   Pros: Good control over trade-offs.
   Cons: Needs multiple runs, hard to choose ϵ.

3. **Tchebycheff (Chebyshev) Scalarization** Minimize the maximum weighted deviation from an ideal point.

$$F(x) = \max_i \{w_i |f_i(x) - z_i^*|\}$$

   Pros: Captures non-convex fronts.
   Cons: Needs ideal point z_i^*.

4. **Min–Max Scalarization** Balance worst normalized objective.

$$\min_x \ \max_i \left(\frac{f_i(x) - f_i^{\min}}{f_i^{\max} - f_i^{\min}} \right)$$

   Pros: No weights, fair balance.
   Cons: Sensitive to min/max values, single solution only.

5. **Goal Programming** Minimize deviation from pre-defined goals.

$$\min \sum_i w_i |f_i(x) - g_i|$$

   Pros: Goal-driven optimization.
   Cons: Requires accurate goals, may not be Pareto-optimal.

Normalized Objectives

Let:

- $f_1(x) = \sum_{i \in D} \sum_{j \in R} x_{ij}$
- $f_2(x) = \frac{1}{|D|} \sum_{i \in D} \left(\sum_{j \in R} x_{ij} - \frac{1}{|D|} \sum_{i \in D} \sum_{j \in R} x_{ij} \right)^2$
- $f_1^{\min}, f_1^{\max}, f_2^{\min}, f_2^{\max}$: estimated minimum and maximum values for each objective

Min–Max Scalarized Objective

$$\min_x \max \{ f_1^{\text{norm}}(x),\ f_2^{\text{norm}}(x) \}$$

$$\min_x \ \max \left\{ 1 - \frac{f_1(x) - f_1^{\min}}{f_1^{\max} - f_1^{\min}}, \quad \frac{f_2(x) - f_2^{\min}}{f_2^{\max} - f_2^{\min}} \right\}$$

This objective avoids prioritizing one goal over the other and penalizes the worse-performing normalized term.

Note: Normalization is used for *scaling*, not limiting. It ensures both objectives are treated fairly on a common scale.

Constraints

C1. **Unique Assignment:** Each rider is assigned to at most one driver:

$$\sum_{i \in D} x_{ij} \leq 1 \quad \forall j \in \{1, \ldots, |R|\}$$

C2. **Capacity Constraint:** Driver cannot exceed vehicle capacity:

$$\sum_{j \in R} x_{ij} \leq n_i \quad \forall i \in \{1, \ldots, |D|\}$$

C3. **Route Deviation Constraint:** Drivers must not exceed their permissible deviation:

$$\Delta_i \leq t_i \cdot |P_i| \quad \forall i \in \{1, \ldots, |D|\}$$

where Δ_i is computed based on the route deviation from serving assigned riders.

Trade-off and Limitation Discussion

When Min–Max Scalarization is Effective

- When a single solution is required.
- When you can estimate bounds $f^{\min}, f^{\max}$ effectively.
- When equal importance is given to both objectives.

When It Fails or Is Suboptimal

- When objectives are non-convex or the trade-off curve is highly non-linear.
- When we want to explore different fairness vs. assignment.
- When normalization ranges are inaccurate or unstable.

Solution

- **If $f_i(x)$ are linear or quadratic (convex):**
  Use **convex optimization solvers** such as **Gurobi**, **CPLEX** for efficient and exact solutions.
- **If $f_i(x)$ are non-convex:**
  Use **global or heuristic optimization methods**, such as **Genetic Algorithms (GA)** since convex solvers cannot guarantee global optimality.

Genetic Algorithm

Which type of chromosome will we use in this problem? A possible solution where each rider is assigned to a driver (or none).
Representation: Follows the style used in genetic algorithms (see *Introduction to Evolutionary Computing* by Eiben & Smith).

1. **Binary Representation**
2. **Integer Representation**
3. **Real-Valued (Floating-Point) Representation**
4. **Permutation Representation**

1. Binary Representation

A binary matrix where each row corresponds to drivers and each column corresponds to a rider.

$$\text{Example of chromosome:} \quad \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \begin{cases} r_1, r_2 \to d_1 \\ r_3, r_4 \to d_2 \end{cases}$$

Crossover: Swap selected rows or columns.

Mutation: Flip a bit (e.g., $0 \leftrightarrow 1$), then enforce one-hot encoding per row.

2. Integer Representation

Binary encoding is suboptimal for discrete variables; for example, evolving a grid path is better represented using integer encoding like {0,1,2,3} for {North, East, South, West}.

Encoding Strategies

A. Rider → Driver Encoding Each rider is assigned directly to a driver. Chromosome is a vector of driver IDs indexed by rider.(Easy to implement)

$$\text{Chromosome A: } [d_1, d_1, d_2, d_2] \Rightarrow \begin{cases} r_1 \to d_1 \\ r_2 \to d_1 \\ r_3 \to d_2 \\ r_4 \to d_2 \end{cases}$$

Crossover: One-point crossover.

$$\text{Parent A: } [d_1, d_1, d_2, d_2], \quad \text{Parent B: } [d_2, d_2, d_1, d_1]$$
$$\text{Child: } [d_1, d_1, d_1, d_1] \quad \text{(may require repair)}$$

Mutation: Randomly reassign a rider to another driver.

B. Driver → Riders Encoding Each driver has a list of assigned riders.

$$\text{Chromosome A: } \begin{cases} d_1 : [r_1, r_2] \\ d_2 : [r_3, r_4] \end{cases}$$

Crossover: Swap riders between driver assignments.

Mutation: Move a rider from one driver to another.

3. Real-Valued (Floating-Point) Representation Ride-sharing problems with discrete entities (drivers, riders, vehicle capacities, assignments) are not naturally continuous, so real-valued chromosomes aren't a direct fit.

4. Permutation Representation Chromosome is a permutation of riders. A decoder assigns each rider to the best-fit driver.
We randomly generate a chromosome as a permutation of riders:

$$\text{Chromosome} = [r_3, r_1, r_4, r_2]]$$

Decoder Logic: We loop through riders and try to assign each to any driver who:

1. Has capacity
2. Can reach rider's source & destination within deviated path.
3. Total path deviation $\leq t_i$ * shortest path

Driver	**Riders**	**Rider Count**
d1	r1, r2	2
d2	r3, r4	2

Crossover: Order Crossover (OX), Cycle Crossover, Edge Crossover, & Partially Mapped Crossover (PMX)

Mutation: Swap two rider positions

Table 1: Comparison of Chromosome Representations for Rider–Driver Assignment

Representation Type	Feasibility Encoding	Natural Fit	Repair Needed	Efficiency	Summary
Binary Matrix	Hard to enforce	Medium	High	Low	Poor scaling and constraint handling
Integer: Rider→Driver	Easy	Direct mapping	Medium	High	**Best trade-off** for this problem
Integer: Driver→Riders	Hard to decode	Good for fixed capacity	High	Medium	Complex crossover/mutation
Real-Valued	No	Poor fit	Total repair	Very low	Inappropriate for discrete assignment
Permutation	Indirect	Good for decoding	Medium	Fast	Great for heuristics, harder to enforce fairness

Genetic Algorithm with Min–Max Scalarization using Integer Representation: Rider → Driver
Chromosome Representation:

$$\text{Chromosome: } [2, 1, -1, 2, 3] \Rightarrow \begin{cases} r_1 \to d_2 \\ r_2 \to d_1 \\ r_3 \to \text{unassigned(-1)} \\ r_4 \to d_2 \\ r_5 \to d_3 \end{cases}$$

Step 1: Initialization

- Set population size N, maximum number of generations G.
- For each chromosome in the population:
  - For each rider:
    - Randomly assign a compatible driver d_i, satisfying:
      - Deviation constraint t_i
      - Capacity constraint n_i
    - Or assign -1 if no feasible driver exists.

Step 2: Fitness Evaluation (Min–Max Scalarization)
For each chromosome:

1. Normalize objectives:

$$f_1^{\text{norm}} = 1 - \frac{f_1(x) - f_1^{\min}}{f_1^{\max} - f_1^{\min}}$$

$$f_2^{\text{norm}} = \frac{f_2(x) - f_2^{\min}}{f_2^{\max} - f_2^{\min}}$$

2. Compute scalarized fitness:

$$F(x) = \max\left(f_1^{\text{norm}}, f_2^{\text{norm}}\right)$$

Step 3: Selection:

Use **tournament selection** based on the scalarized fitness $F(x)$, where a **lower** fitness value indicates a **better** solution (minimization).

Step 4: Crossover

- Use **one-point crossover** to generate offspring from two parent chromosomes.
- Randomly select a crossover point $c \in [1, n-1]$, where n is the chromosome length (number of riders).
- The first part (genes before point c) is taken from Parent A, and the remaining part is taken from Parent B.

Example:

$$\begin{aligned} \text{Parent A:} \quad & [2,\ 1,\ -1,\ 2,\ 3] \\ \text{Parent B:} \quad & [1,\ 3,\ 2,\ -1,\ 1] \\ \text{Crossover point:} \quad & c = 2 \\ \text{Child:} \quad & [2,\ 1,\ 2,\ -1,\ 1] \end{aligned}$$

Step 5: Mutation

- For each gene (rider), apply mutation with probability p_m.
- If selected for mutation:
  - Reassign the rider to a different feasible driver (one satisfying constraints), or
  - Set to -1 if no feasible assignment exists.
- **Post-Mutation Repair:**
  - Ensure that no driver exceeds:
    * Capacity constraint n_i
    * Route deviation constraint $t_i \cdot |P_i|$, where $|P_i|$ is the number of riders assigned to driver i

Example:

$$\begin{aligned} \text{Original chromosome:} \quad & [2,\ 1,\ -1,\ 2,\ 3] \\ \text{Mutation point:} \quad & \text{Gene 3 (rider 3)} \\ \text{Before mutation:} \quad & \text{rider } 3 \to \text{unassigned} \\ \text{After mutation:} \quad & \text{rider } 3 \to d_1 \\ \text{Mutated chromosome:} \quad & [2,\ 1,\ \mathbf{1},\ 2,\ 3] \end{aligned}$$

Repair Step:

- Suppose driver d_1 has capacity $n_1 = 2$ and is now assigned to riders 2 and 3.
- If route deviation constraint is violated (e.g., $t_1 \cdot |P_1|$ exceeded), revert rider 3 back to -1.

Final chromosome (after repair, if needed):

$$[2, 1, -1, 2, 3] \quad \text{(if rider 3 was removed due to constraint violation)}$$

Step 6: Elitism & Replacement

- Carry over the top e elite chromosomes (selection same as step 3) directly to the next generation.
- Fill the remaining $N - e$ slots in the population with newly generated offspring.

Step 7: Termination

- Repeat Steps 2–6 for G generations or until a convergence criterion is met.
- Return the solution (chromosome) with the minimum scalarized fitness $F(x)$.