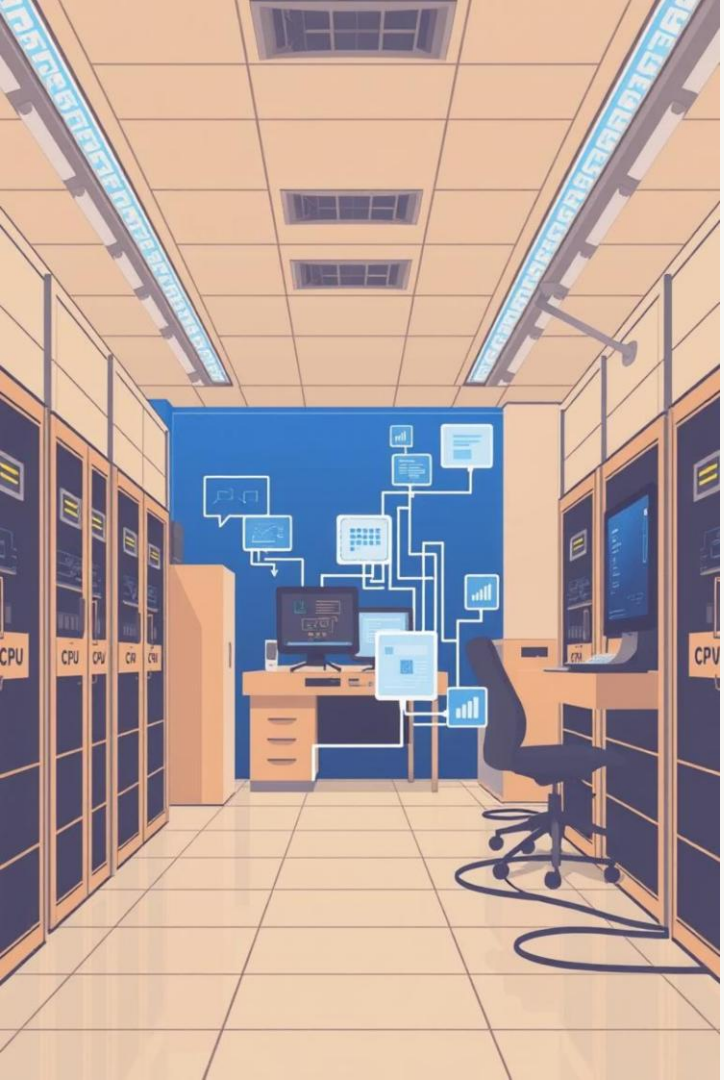
A Modular Genetic Algorithms Library in Java

Designing a reusable, extensible framework for solving optimization problems through configurable genetic operators and representations.



CPU Job Scheduling: The Challenge

The Problem

Given jobs with arrival times and burst times, find the optimal execution order that minimizes average waiting time and turnaround time.

Why Genetic Algorithms?

NP-hard problem with massive search space. GAs efficiently explore solutions without requiring precise mathematical formulation.

Chromosome Representation

Each chromosome encodes a job permutation representing execution order.

Example Chromosome

[P1, P3, P2, P4]

Gene Structure

Each gene = one job

Representation Type

JOB permutation

Fitness Function Design

Quality measured by inverse of weighted waiting and turnaround times, with penalties for invalid schedules.

$$f = \frac{1}{1 + (1.5 \times \text{Avg Waiting}) + \text{Avg Turnaround}}$$

Duplicate Jobs

×0.5 penalty

Early Start

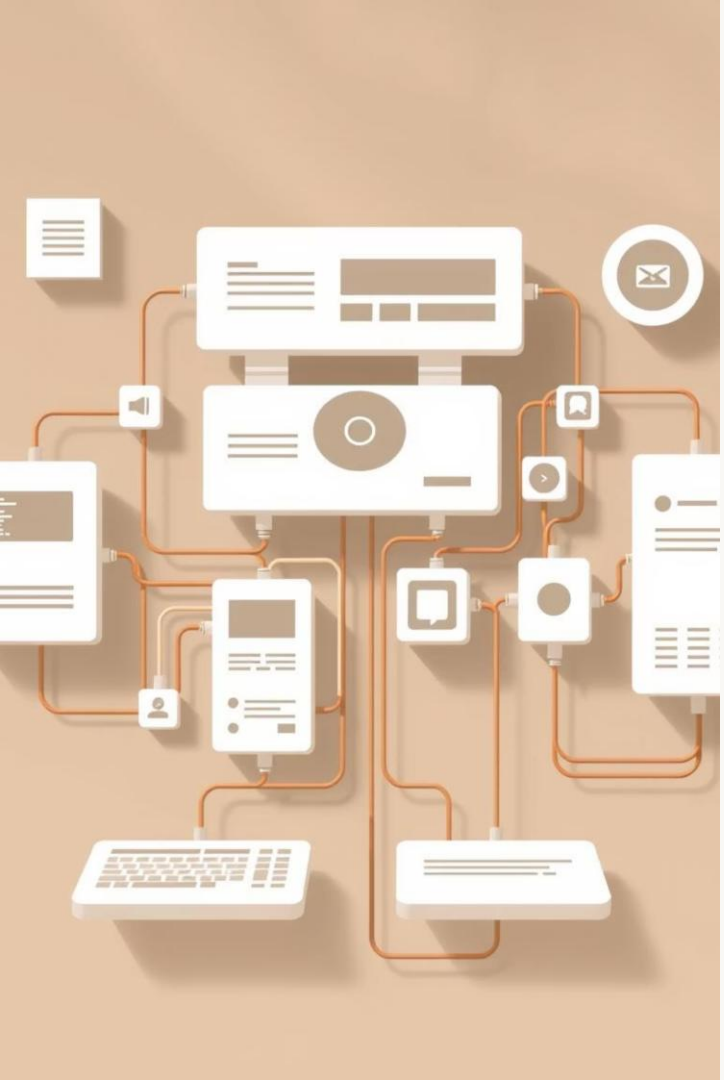
×0.3 penalty

High Fitness

Lower wait and turnaround times

System Architecture

Layered design with separation of concerns across core, engine, operators, and case studies.



SOLID Design Principles

Framework built on software engineering best practices enabling independent testing, reuse, and extension.

- **Single Responsibility**

  Each operator has one focused role

- **Open/Closed**

  Add operators without changing existing code

- **Liskov Substitution**

  Any operator swappable through interfaces

- **Dependency Inversion**

  Engine depends on abstractions, not concrete classes

Genetic Operators: Selection, Crossover, Mutation

Tournament Selection

Controlled pressure, maintains diversity, robust in uneven landscapes

Order Crossover (OX)

Preserves job order, prevents duplicates, ideal for permutations

Swap Mutation

Safe for permutations, encourages local exploration

Replacement Strategy: Steady-State

Why Steady-State?

- Gradually replaces worst individuals
- Maintains strong continuity
- Prevents elite loss early on
- Memory-efficient and stable

Evolution Pipeline & Performance Tracking

01

Initialize Population

02

Evaluate Fitness

03

Selection & Crossover

04

Mutation & Replacement

05

Repeat Until Convergence

PerformanceMetrics tracks best/average fitness, runtime, and memory usage. Auto-exported to CSV with JavaFX visualization.

Extensible Framework for Real-World Optimization

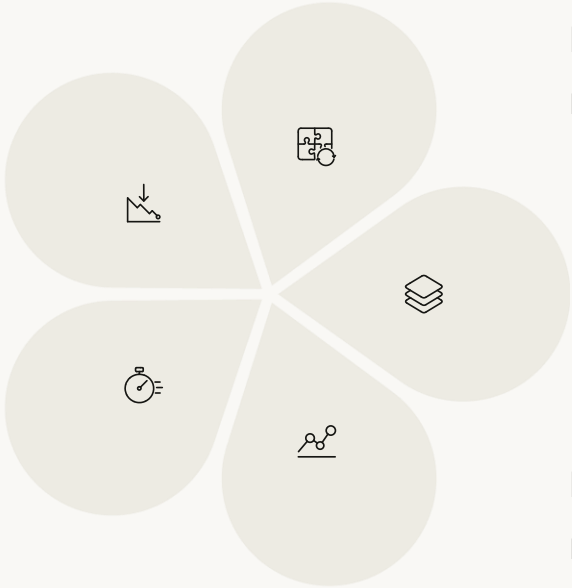
Modular architecture enables rapid deployment across diverse optimization domains. CLI configurator allows dynamic operator selection and parameter tuning for experimentation and testing.

CLI Configuration

Dynamic operator and parameter selection

Production-Ready

High-performance optimization engine



Modular Design

Independent testing and reuse

Extensible Framework

Add case studies without core changes

Performance Metrics

Real-time visualization and analysis