Evolutional Computing Repository

Description

This repository contains the implementation of genetic algorithms for optimization, including selection, crossover, and mutation methods. The main script, `MainOptimizationScript.py`, is responsible for executing the optimization process and evaluating the algorithm's performance. Additional scripts and libraries provide modular implementations of key components.

Repository Structure

```
EvolutionalComputing/
├── MainOptimizationScript.py       # Main script for optimization
├── Playground.py                   # Auxiliary script for testing and
experimentation
├── Experiments_1A/                 # Scripts for running specific experiments
│   ├── ExperimentPopulationSize.py # Experiment: Varying population sizes
│   ├── ExperimentCrossoverRate.py  # Experiment: Varying crossover rates
│   ├── ExperimentMutationRate.py   # Experiment: Varying mutation rates
│   └── ExperimentElitismProportion.py # Experiment: Varying elitism proportions
├── Library/                        # Library with selection, crossover, and
mutation methods
│   ├── CrossoverMethods.py         # Crossover methods
│   ├── MutationMethods.py          # Mutation methods
│   ├── SelectionMethods.py         # Selection methods
│   └── __pycache__/                # Compiled files
├── Results/                        # Directory for storing results
└── _compiled/                      # Directory for compiled files
```

Main Scripts

MainOptimizationScript.py

This is the main script that implements the genetic algorithm. It includes:

- **Algorithm Configuration**: Parameters such as population size, number of generations, selection, crossover, and mutation methods.
- **Fitness Functions**: Implementation of fitness functions like Base, Ackley, Drop-Wave, and Levi.
- **Performance Metrics**: Success rate, best fitness, population diversity, and execution time.
- **Stopping Criteria**: Configurable stopping criteria:
  - Fixed number of generations (`GENERATION_COUNT`).
  - Discovery of a solution with a desired fitness (`TARGET_FITNESS`).
  - No improvement over a specified number of generations (`NO_IMPROVEMENT_LIMIT`).
- **Visualizations**: Convergence, population diversity, and optimal points distribution plots.

Key Methods

- `evaluate_fitness`: Evaluates the fitness of a chromosome based on the selected function.
- `single_optimization`: Executes a single optimization.
- `multiple_optimization`: Executes multiple optimizations and calculates aggregated metrics.
- `elitism_optimization`: Implements the elitism method for population evolution.
- `maintain_diversity`: Applies strategies to maintain population diversity.
- `plot_convergence_curve`: Plots the convergence curve.
- `plot_population_diversity`: Plots the population diversity.
- `plot_optimal_points`: Plots the distribution of optimal points.
- `save_results`: Saves results, configuration, and visualizations to a timestamped folder.

Playground.py

This script is used for testing and experimenting with different configurations and methods of the genetic algorithm. It allows developers to:

- Test new fitness functions.
- Experiment with different selection, crossover, and mutation methods.
- Debug and validate the behavior of the genetic algorithm components.

Experiments_1A/

ExperimentPopulationSize.py

This script evaluates the impact of varying **population sizes** on the algorithm's performance. It analyzes how population size affects convergence, diversity, and execution time.

ExperimentCrossoverRate.py

This script analyzes the effect of varying **crossover rates** on the algorithm's performance. It evaluates how crossover impacts diversity and convergence.

ExperimentMutationRate.py

This script analyzes the effect of varying **mutation rates** on the algorithm's performance. It evaluates how mutation impacts diversity and convergence.

ExperimentElitismProportion.py

This script evaluates the impact of different **elitism proportions** on the optimization process. It helps determine the most efficient proportion of elites for specific problems.

Library/

SelectionMethods.py

Implements various selection methods for choosing parents in the genetic algorithm:

- **Tournament Selection**: Selects the best individual from a randomly chosen subset of the population.
- **Roulette Wheel Selection**: Selects individuals with a probability proportional to their fitness.

CrossoverMethods.py

Implements crossover methods for combining parent chromosomes to produce offspring:

- **Single-Point Crossover**: Splits parent chromosomes at a random point and swaps segments.
- **Arithmetic Crossover**: Combines parent chromosomes using a weighted average.

MutationMethods.py

Implements mutation methods for introducing random changes to chromosomes:

- **Random Mutation on Individual Genes**: Randomly alters genes in a chromosome based on a mutation rate.

How to Use

1. Configure the parameters in the `MainOptimizationScript.py` file.
2. Run the script to perform optimizations:

```
python MainOptimizationScript.py
```

3. To run specific experiments, navigate to the `Experiments_1A` folder and execute the desired script:

```
python Experiments_1A/ExperimentPopulationSize.py
```

4. Visualize the results in the generated plots and metrics displayed in the console.

Requirements

- Python 3.11 or higher
- Required libraries:
  - `numpy`
  - `matplotlib`
  - `scipy`

Install the dependencies with:

```
pip install numpy matplotlib scipy
```

Contact

For questions or suggestions, contact Gabriel Fernandes.