Fuzzy Logic Controlled Genetic Algorithm (FLC-GA) designed to dynamically adjust genetic algorithm parameters (mutation and crossover rates) to improve optimization efficiency and adaptiveness. It combines a genetic algorithm (GA) with a fuzzy logic controller that uses problem-specific characteristics—namely, diversity and convergence rate—to control the GA's search behavior in a way that balances exploration and exploitation across generations.

Project Components Genetic Algorithm (GA) Basics: The GA is an optimization algorithm inspired by natural selection. It works with a population of candidate solutions (individuals) that evolve over multiple generations. Each individual is evaluated by a fitness function, which assesses how good that solution is with respect to the optimization goal. In each generation, a selection of the best-performing individuals is combined and mutated to produce the next generation, gradually improving the population toward an optimal solution.

Adaptation with Fuzzy Logic: Traditional GAs use fixed values for mutation rate (probability of altering individual genes) and crossover rate (probability of combining parts of two parent solutions). However, setting these parameters is challenging, as the best values may change depending on the search stage. To address this, the fuzzy logic controller dynamically adjusts the mutation and crossover rates based on two key metrics:

Diversity: A measure of how different the individuals in the population are from each other. Higher diversity suggests the algorithm is still exploring various solutions, while lower diversity indicates that solutions are converging (becoming more similar).

Convergence Rate: Indicates how close the population is to reaching an optimal solution. Early generations require exploration (higher mutation and/or crossover rates), while later generations benefit from exploitation, or focusing on fine-tuning the best solutions (lower mutation rates).

Fuzzy Logic Controller (FLC): The FLC takes the diversity and convergence rate as inputs to generate output values for mutation and crossover rates. Here’s how it works:

Fuzzy Variables: Diversity and convergence rate are described using linguistic terms (e.g., "low," "high") with associated membership functions. Similarly, mutation rate and crossover rate are also defined in terms like "low" or "high."

Fuzzy Rules: Rules are formulated based on how different levels of diversity and convergence rate should impact the mutation and crossover rates. For instance:

"If diversity is low and convergence is fast, then mutation rate should be low, and crossover rate should be high." "If diversity is high and convergence is slow, then mutation rate should be high, and crossover rate should be low." The FLC interprets the diversity and convergence metrics in each generation to infer the best mutation and crossover rates.

Optimization Process:

Each generation of the GA, individuals are selected based on fitness, and the FLC adjusts the mutation and crossover rates based on current diversity and convergence values. Selection, Crossover, and Mutation steps in the GA are performed with the rates recommended by the FLC, leading to a more adaptive search process that balances exploration (broad search of solution space) and exploitation (fine-tuning of good solutions). Result: Over time, this hybrid approach helps the GA avoid common pitfalls, such as premature convergence (getting stuck in a

suboptimal solution) and excessive exploration (wasting resources on too much random searching). By adapting parameters in real-time, the FLC-GA can efficiently find optimal or near-optimal solutions.

Project Applications

This FLC-GA framework is useful for complex optimization problems where fixed GA parameters may not yield efficient or optimal results. It's particularly suitable for:

Machine Learning Hyperparameter Tuning: FLC-GA can dynamically adjust algorithm settings based on training progress. Complex Engineering Design: For problems with a large design space, such as aerodynamic design or structural optimization. Logistics and Scheduling: The algorithm can optimize routes, resource allocation, or schedules with changing constraints and requirements.

```
pip install scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl.metadata (2.6
kB)
Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl (920 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 920.8/920.8 kB 9.4 MB/s eta
0:00:00
```

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import random

# 1. Define fuzzy variables
diversity = ctrl.Antecedent(np.arange(0, 1.1, 0.1), 'diversity')
convergence = ctrl.Antecedent(np.arange(0, 1.1, 0.1), 'convergence')
mutation_rate = ctrl.Consequent(np.arange(0, 1.1, 0.1),
'mutation_rate')
crossover_rate = ctrl.Consequent(np.arange(0, 1.1, 0.1),
'crossover_rate')

# 2. Define membership functions for fuzzy variables
diversity['low'] = fuzz.trimf(diversity.universe, [0, 0, 0.5])
diversity['high'] = fuzz.trimf(diversity.universe, [0.5, 1, 1])
convergence['slow'] = fuzz.trimf(convergence.universe, [0, 0, 0.5])
convergence['fast'] = fuzz.trimf(convergence.universe, [0.5, 1, 1])
mutation_rate['low'] = fuzz.trimf(mutation_rate.universe, [0, 0, 0.5])
mutation_rate['high'] = fuzz.trimf(mutation_rate.universe, [0.5, 1,
1])
crossover_rate['low'] = fuzz.trimf(crossover_rate.universe, [0, 0,
0.5])
crossover_rate['high'] = fuzz.trimf(crossover_rate.universe, [0.5, 1,
1])

# 3. Define fuzzy rules with additional coverage
```

```
rule1 = ctrl.Rule(diversity['low'] & convergence['fast'],
(mutation_rate['low'], crossover_rate['high']))
rule2 = ctrl.Rule(diversity['high'] & convergence['slow'],
(mutation_rate['high'], crossover_rate['low']))
rule3 = ctrl.Rule(diversity['low'] & convergence['slow'],
(mutation_rate['high'], crossover_rate['low']))
rule4 = ctrl.Rule(diversity['high'] & convergence['fast'],
(mutation_rate['low'], crossover_rate['high']))
rule5 = ctrl.Rule(diversity['low'] & convergence['slow'],
(mutation_rate['high'], crossover_rate['high']))
rule6 = ctrl.Rule(diversity['high'] & convergence['slow'],
(mutation_rate['low'], crossover_rate['low']))

# Control system creation and simulation
fuzzy_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5,
rule6])
fuzzy_sim = ctrl.ControlSystemSimulation(fuzzy_ctrl)

# 4. Define a simple fitness function (depends on the problem)
def fitness(solution):
    # Example: Count the number of '1's in a binary solution for
demonstration
    return sum(solution)

# 5. Initialize population
def create_population(size, solution_length):
    return [np.random.randint(0, 2, solution_length) for _ in
range(size)]

# Genetic algorithm parameters
population_size = 50
generations = 20
solution_length = 10  # Length of each solution

# GA main loop
population = create_population(population_size, solution_length)
for gen in range(generations):
    # Calculate diversity and convergence rate
    fitness_values = [fitness(individual) for individual in
population]
    avg_fitness = np.mean(fitness_values)
    diversity_value = np.std(fitness_values) / avg_fitness if
avg_fitness > 0 else 0  # Diversity metric
    convergence_value = (generations - gen) / generations  # Example
convergence rate

    # Set inputs for the fuzzy controller
    fuzzy_sim.input['diversity'] = diversity_value
    fuzzy_sim.input['convergence'] = convergence_value
    fuzzy_sim.compute()
```

```
    # Check if outputs are present; use default values if not
    mutation_chance = fuzzy_sim.output.get('mutation_rate', 0.1)  #
Default mutation rate
    crossover_chance = fuzzy_sim.output.get('crossover_rate', 0.7)  #
Default crossover rate

    print(f"Generation {gen+1}, Diversity: {diversity_value:.2f},
Convergence: {convergence_value:.2f}")
    print(f"Mutation Rate: {mutation_chance:.2f}, Crossover Rate:
{crossover_chance:.2f}")

    # Selection (simple tournament selection)
    selected_parents = random.choices(population, k=population_size)

    # Crossover
    next_population = []
    for i in range(0, population_size, 2):
        parent1, parent2 = selected_parents[i], selected_parents[(i+1)
% population_size]
        if random.random() < crossover_chance:
            crossover_point = random.randint(1, solution_length - 1)
            child1 = np.concatenate([parent1[:crossover_point],
parent2[crossover_point:]])
            child2 = np.concatenate([parent2[:crossover_point],
parent1[crossover_point:]])
            next_population.extend([child1, child2])
        else:
            next_population.extend([parent1, parent2])

    # Mutation
    for individual in next_population:
        for gene in range(solution_length):
            if random.random() < mutation_chance:
                individual[gene] = 1 - individual[gene]  # Flip bit

    # Update population
    population = next_population

# Final output after all generations
best_solution = max(population, key=fitness)
print("Best solution found:", best_solution)
print("Best solution fitness:", fitness(best_solution))
```

```
Generation 1, Diversity: 0.28, Convergence: 1.00
Mutation Rate: 0.20, Crossover Rate: 0.80
Generation 2, Diversity: 0.29, Convergence: 0.95
Mutation Rate: 0.20, Crossover Rate: 0.80
Generation 3, Diversity: 0.27, Convergence: 0.90
Mutation Rate: 0.20, Crossover Rate: 0.80
```

```
Generation 4, Diversity: 0.29, Convergence: 0.85
Mutation Rate: 0.20, Crossover Rate: 0.80
Generation 5, Diversity: 0.30, Convergence: 0.80
Mutation Rate: 0.20, Crossover Rate: 0.80
Generation 6, Diversity: 0.31, Convergence: 0.75
Mutation Rate: 0.21, Crossover Rate: 0.79
Generation 7, Diversity: 0.32, Convergence: 0.70
Mutation Rate: 0.21, Crossover Rate: 0.79
Generation 8, Diversity: 0.36, Convergence: 0.65
Mutation Rate: 0.22, Crossover Rate: 0.78
Generation 9, Diversity: 0.29, Convergence: 0.60
Mutation Rate: 0.23, Crossover Rate: 0.77
Generation 10, Diversity: 0.27, Convergence: 0.55
Mutation Rate: 0.24, Crossover Rate: 0.76
Generation 11, Diversity: 0.27, Convergence: 0.50
Mutation Rate: 0.10, Crossover Rate: 0.70
Generation 12, Diversity: 0.30, Convergence: 0.45
Mutation Rate: 0.76, Crossover Rate: 0.50
Generation 13, Diversity: 0.33, Convergence: 0.40
Mutation Rate: 0.77, Crossover Rate: 0.50
Generation 14, Diversity: 0.24, Convergence: 0.35
Mutation Rate: 0.79, Crossover Rate: 0.50
Generation 15, Diversity: 0.29, Convergence: 0.30
Mutation Rate: 0.80, Crossover Rate: 0.50
Generation 16, Diversity: 0.29, Convergence: 0.25
Mutation Rate: 0.80, Crossover Rate: 0.50
Generation 17, Diversity: 0.35, Convergence: 0.20
Mutation Rate: 0.79, Crossover Rate: 0.50
Generation 18, Diversity: 0.27, Convergence: 0.15
Mutation Rate: 0.80, Crossover Rate: 0.50
Generation 19, Diversity: 0.32, Convergence: 0.10
Mutation Rate: 0.79, Crossover Rate: 0.50
Generation 20, Diversity: 0.31, Convergence: 0.05
Mutation Rate: 0.79, Crossover Rate: 0.50
Best solution found: [0 1 1 1 1 0 1 1 1 0]
Best solution fitness: 7
```