```
import random

def genetic_algorithm(population_size, mutation_rate,
crossover_rate, num_variables):
    # Generate initial population
    initial_population = [[random.randint(0, 10) for _
in range(num_variables)] for _ in range(population_size
)]
    print("Initial Population:")
    for chromosome in initial_population:
        print(chromosome)

    # Loop until optimal solution found (fitness of 0)
    population = initial_population
    while True:
        fitness_values = [fitness(chromosome) for
chromosome in population]
        best_chromosome = population[fitness_values.
index(min(fitness_values))]
        if min(fitness_values) == 0:
            print("\nFinal Population:")
            for chromosome in population:
                print(chromosome)
            return best_chromosome

        # Selection (replace with a simpler method if
needed)
        selected_population = random.choices(population
, fitness_values, k=population_size)

        # Crossover (replace with a basic mechanism if
needed)
        next_generation = []
        for _ in range(population_size // 2):
            parent1, parent2 = random.sample(
selected_population, 2)
            child1, child2 = crossover(parent1, parent2
, crossover_rate)
            next_generation.extend([child1, child2])

        # Mutation
        for individual in next_generation:
            mutation(individual, mutation_rate)

        population = next_generation

def fitness(variables):
    a, b, c, d = variables
    return abs((a + 2 * b + 3 * c + 4 * d) - 30)

# Simplified crossover (can be replaced with random
selection)
def crossover(parent1, parent2, crossover_rate):
    if random.random() < crossover_rate:
        crossover_point = random.randint(1, len(parent1
) - 1)
        return parent1[:crossover_point] + parent2[
crossover_point:], parent2[:crossover_point] + parent1[
crossover_point:]
    return parent1, parent2

# Simplified mutation (can be replaced with random
value assignment)
def mutation(individual, mutation_rate):
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            individual[i] = random.randint(0, 10)

def main():
    population_size = 6
    mutation_rate = 0.1  # Example values
    crossover_rate = 0.7
    num_variables = 4
    best_chromosome = genetic_algorithm(population_size
, mutation_rate, crossover_rate, num_variables)
    print("\nBest solution:", best_chromosome)

main()
```