```python
import random

def fitness(x):
    return x**2

def initialize_population(pop_size, low, high):
    return [random.uniform(low, high) for _ in range(pop_size)]

def selection(population, fitness_values):
    total_fitness = sum(fitness_values)
    selection_probs = [f / total_fitness for f in fitness_values]
    return random.choices(population, weights=selection_probs, k=2)

def crossover(parent1, parent2):
    alpha = random.random()
    offspring1 = alpha * parent1 + (1 - alpha) * parent2
    offspring2 = alpha * parent2 + (1 - alpha) * parent1
    return offspring1, offspring2

def mutate(individual, mutation_rate, low, high):
    if random.random() < mutation_rate:
        return random.uniform(low, high)
    return individual

def genetic_algorithm(pop_size, generations, low, high, mutation_rate, crossover_rate):
    population = initialize_population(pop_size, low, high)
    best_solution = None
    best_fitness = float('-inf')

    for generation in range(generations):
        fitness_values = [fitness(ind) for ind in population]

        max_fitness = max(fitness_values)
        if max_fitness > best_fitness:
            best_fitness = max_fitness
            best_solution = population[fitness_values.index(max_fitness)]

        new_population = []
        while len(new_population) < pop_size:
            parent1, parent2 = selection(population, fitness_values)

            if random.random() < crossover_rate:
                offspring1, offspring2 = crossover(parent1, parent2)
            else:
                offspring1, offspring2 = parent1, parent2

            offspring1 = mutate(offspring1, mutation_rate, low, high)
            offspring2 = mutate(offspring2, mutation_rate, low, high)

            new_population.extend([offspring1, offspring2])

        population = new_population[:pop_size]

        print(f"Generation {generation+1}: Best fitness = {best_fitness:.4f}, Best solution = {best_solution:.4f}")

    print(f"\nBest solution found: x = {best_solution:.4f}, f(x) = {best_fitness:.4f}")

population_size = 100
num_generations = 50
x_range_low = -10
x_range_high = 10
mutation_rate = 0.1
crossover_rate = 0.7

genetic_algorithm(population_size, num_generations, x_range_low, x_range_high, mutation_rate, crossover_rate)
```

```
Generation 1: Best fitness = 99.2016, Best solution = 9.9600
Generation 2: Best fitness = 99.2016, Best solution = 9.9600
Generation 3: Best fitness = 99.2016, Best solution = 9.9600
Generation 4: Best fitness = 99.2016, Best solution = 9.9600
Generation 5: Best fitness = 99.2016, Best solution = 9.9600
Generation 6: Best fitness = 99.2016, Best solution = 9.9600
Generation 7: Best fitness = 99.2016, Best solution = 9.9600
Generation 8: Best fitness = 99.2016, Best solution = 9.9600
Generation 9: Best fitness = 99.2016, Best solution = 9.9600
Generation 10: Best fitness = 99.2016, Best solution = 9.9600
Generation 11: Best fitness = 99.2016, Best solution = 9.9600
Generation 12: Best fitness = 99.2016, Best solution = 9.9600
Generation 13: Best fitness = 99.2016, Best solution = 9.9600
Generation 14: Best fitness = 99.2016, Best solution = 9.9600
Generation 15: Best fitness = 99.7462, Best solution = -9.9873
Generation 16: Best fitness = 99.7462, Best solution = -9.9873
Generation 17: Best fitness = 99.7462, Best solution = -9.9873
Generation 18: Best fitness = 99.7462, Best solution = -9.9873
Generation 19: Best fitness = 99.7462, Best solution = -9.9873
Generation 20: Best fitness = 99.7462, Best solution = -9.9873
Generation 21: Best fitness = 99.7462, Best solution = -9.9873
Generation 22: Best fitness = 99.7462, Best solution = -9.9873
Generation 23: Best fitness = 99.7462, Best solution = -9.9873
Generation 24: Best fitness = 99.7462, Best solution = -9.9873
Generation 25: Best fitness = 99.7462, Best solution = -9.9873
Generation 26: Best fitness = 99.7462, Best solution = -9.9873
Generation 27: Best fitness = 99.7462, Best solution = -9.9873
Generation 28: Best fitness = 99.7462, Best solution = -9.9873
Generation 29: Best fitness = 99.7462, Best solution = -9.9873
Generation 30: Best fitness = 99.7462, Best solution = -9.9873
Generation 31: Best fitness = 99.7462, Best solution = -9.9873
Generation 32: Best fitness = 99.7462, Best solution = -9.9873
Generation 33: Best fitness = 99.7462, Best solution = -9.9873
Generation 34: Best fitness = 99.7462, Best solution = -9.9873
Generation 35: Best fitness = 99.7462, Best solution = -9.9873
Generation 36: Best fitness = 99.7462, Best solution = -9.9873
Generation 37: Best fitness = 99.7462, Best solution = -9.9873
Generation 38: Best fitness = 99.7462, Best solution = -9.9873
Generation 39: Best fitness = 99.7462, Best solution = -9.9873
Generation 40: Best fitness = 99.7462, Best solution = -9.9873
Generation 41: Best fitness = 99.7462, Best solution = -9.9873
Generation 42: Best fitness = 99.7462, Best solution = -9.9873
Generation 43: Best fitness = 99.7462, Best solution = -9.9873
Generation 44: Best fitness = 99.7462, Best solution = -9.9873
Generation 45: Best fitness = 99.7462, Best solution = -9.9873
Generation 46: Best fitness = 99.7462, Best solution = -9.9873
Generation 47: Best fitness = 99.7462, Best solution = -9.9873
Generation 48: Best fitness = 99.7462, Best solution = -9.9873
Generation 49: Best fitness = 99.7462, Best solution = -9.9873
Generation 50: Best fitness = 99.7462, Best solution = -9.9873

Best solution found: x = -9.9873, f(x) = 99.7462
```