Optimization via Gene Expression Algorithms

```
import numpy as np


# Problem definition
def objective_function(x, y):
    return x**2 + y**2  # Example function to minimize


# Initialize parameters
population_size = 100
num_genes = 2  # For (x, y) problem
mutation_rate = 0.1
crossover_rate = 0.7
num_generations = 50
gene_bounds = (-10, 10)  # Range for each gene (x, y)


# Helper functions
def initialize_population():
    return np.random.uniform(gene_bounds[0], gene_bounds[1], (population_size, num_genes))


def evaluate_fitness(population):
    return np.array([objective_function(ind[0], ind[1]) for ind in population])


def select_parents(population, fitness):
    probabilities = 1 / (fitness + 1e-6)  # Convert fitness to probabilities
    probabilities /= probabilities.sum()
    indices = np.random.choice(np.arange(population_size), size=population_size, p=probabilities)
    return population[indices]


def crossover(parent1, parent2):
    if np.random.rand() < crossover_rate:
        point = np.random.randint(1, num_genes)
```

```
        child1 = np.concatenate((parent1[:point], parent2[point:]))
        child2 = np.concatenate((parent2[:point], parent1[point:]))
        return child1, child2
    return parent1, parent2

def mutate(individual):
    for i in range(num_genes):
        if np.random.rand() < mutation_rate:
            individual[i] += np.random.uniform(-1, 1)
            individual[i] = np.clip(individual[i], gene_bounds[0], gene_bounds[1])
    return individual

# Main GEA process
def gene_expression_algorithm():
    population = initialize_population()
    best_solution = None
    best_fitness = float('inf')

    for generation in range(num_generations):
        fitness = evaluate_fitness(population)
        if fitness.min() < best_fitness:
            best_fitness = fitness.min()
            best_solution = population[fitness.argmin()]

        parents = select_parents(population, fitness)
        offspring = []

        for i in range(0, population_size, 2):
            parent1, parent2 = parents[i], parents[i + 1]
            child1, child2 = crossover(parent1, parent2)
            offspring.append(mutate(child1))
```

```python
            offspring.append(mutate(child2))

        population = np.array(offspring)

        print(f"Generation {generation + 1}: Best Fitness = {best_fitness:.5f}, Best Solution = 
{best_solution}")

    return best_solution, best_fitness

# Run the algorithm
best_solution, best_fitness = gene_expression_algorithm()
print("\nBest Solution Found:")
print(f"Solution: {best_solution}, Fitness: {best_fitness:.5f}")
```

```
Generation 10: Best Fitness = 0.00282, Best Solution = [-0.05287625 -0.00484865]
Generation 11: Best Fitness = 0.00282, Best Solution = [-0.05287625 -0.00484865]
Generation 12: Best Fitness = 0.00282, Best Solution = [-0.05287625 -0.00484865]
Generation 13: Best Fitness = 0.00043, Best Solution = [-0.02010828 -0.00484865]
Generation 14: Best Fitness = 0.00043, Best Solution = [-0.02010828 -0.00484865]
Generation 15: Best Fitness = 0.00043, Best Solution = [-0.02010828 -0.00484865]
Generation 16: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 17: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 18: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 19: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 20: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 21: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 22: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 23: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 24: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 25: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 26: Best Fitness = 0.00008, Best Solution = [ 0.00727875 -0.00484865]
Generation 27: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 28: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 29: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 30: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 31: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 32: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 33: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 34: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 35: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 36: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 37: Best Fitness = 0.00007, Best Solution = [0.00727875 0.0046719 ]
Generation 38: Best Fitness = 0.00006, Best Solution = [ 0.00727875 -0.00304049]
Generation 39: Best Fitness = 0.00006, Best Solution = [ 0.00727875 -0.00304049]
Generation 40: Best Fitness = 0.00006, Best Solution = [ 0.00727875 -0.00304049]
Generation 41: Best Fitness = 0.00006, Best Solution = [ 0.00727875 -0.00304049]
Generation 42: Best Fitness = 0.00006, Best Solution = [ 0.00727875 -0.00304049]
Generation 43: Best Fitness = 0.00006, Best Solution = [ 0.00727875 -0.00304049]
Generation 44: Best Fitness = 0.00006, Best Solution = [ 0.00727875 -0.00304049]
Generation 45: Best Fitness = 0.00006, Best Solution = [-0.0056377  -0.00484865]
Generation 46: Best Fitness = 0.00006, Best Solution = [-0.0056377  -0.00484865]
Generation 47: Best Fitness = 0.00006, Best Solution = [-0.0056377  -0.00484865]
Generation 48: Best Fitness = 0.00006, Best Solution = [-0.0056377  -0.00484865]
Generation 49: Best Fitness = 0.00006, Best Solution = [-0.0056377  -0.00484865]
Generation 50: Best Fitness = 0.00004, Best Solution = [-0.0056377  -0.00304049]

Best Solution Found:
Solution: [-0.0056377  -0.00304049], Fitness: 0.00004
```