```python
import numpy as np

def fitness_function(x):
    return x**2

population_size = 40
mutation_rate = 0.1
crossover_rate = 0.7
num_generations = 5
x_range = (-10, 10)

def initialize_population(size, x_range):
    return np.random.uniform(x_range[0], x_range[1], size)

def evaluate_fitness(population):
    return fitness_function(population)

def select_parents(population, fitness):
    probabilities = fitness / fitness.sum()
    parents_indices = np.random.choice(np.arange(len(population)), size=2, p=probabilities)
    return population[parents_indices]

def crossover(parent1, parent2):
    if np.random.rand() < crossover_rate:
        alpha = np.random.rand()
        offspring1 = alpha * parent1 + (1 - alpha) * parent2
        offspring2 = alpha * parent2 + (1 - alpha) * parent1
        return offspring1, offspring2
    return parent1, parent2

def mutate(offspring):
    if np.random.rand() < mutation_rate:
        return np.random.uniform(x_range[0], x_range[1])
    return offspring

def genetic_algorithm():
    population = initialize_population(population_size, x_range)
    for generation in range(num_generations):
        fitness = evaluate_fitness(population)
        new_population = []
        for _ in range(population_size // 2):
            parent1, parent2 = select_parents(population, fitness)
            offspring1, offspring2 = crossover(parent1, parent2)
            new_population.append(mutate(offspring1))
            new_population.append(mutate(offspring2))
        population = np.array(new_population)
        best_fitness_index = np.argmax(fitness)
        best_solution = population[best_fitness_index]
        best_value = fitness[best_fitness_index]
        print(f"Generation {generation + 1}: Best Solution = {best_solution}, Best Fitness = {best_value}")
    best_fitness_index = np.argmax(fitness)
    best_solution = population[best_fitness_index]
    best_value = fitness[best_fitness_index]
    return best_solution, best_value

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")

best_solution, best_value = genetic_algorithm()
print(f"Best Solution Found: x = {best_solution}, f(x) = {best_value}")
```

```
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012
Generation 1: Best Solution = -1.5600468045759253, Best Fitness = 98.58395329865364
Generation 2: Best Solution = 0.2113178131917497, Best Fitness = 98.58395329865364
Generation 3: Best Solution = 9.928945225886466, Best Fitness = 98.58395329865364
Generation 4: Best Solution = -9.198951729828826, Best Fitness = 98.58395329865364
Generation 5: Best Solution = -8.46777409147688, Best Fitness = 93.64830237425143
Best Solution Found: x = -8.46777409147688, f(x) = 93.64830237425143
```

```python
import random

def fitness_function(x):
    return -x ** 2

class Particle:
    def __init__(self, min_x, max_x):
        self.position = random.uniform(min_x, max_x)
        self.velocity = random.uniform(-1, 1)
        self.best_position = self.position
        self.best_fitness = fitness_function(self.position)

    def update_velocity(self, global_best_position, inertia, cognitive, social):
        r1, r2 = random.random(), random.random()
        cognitive_velocity = cognitive * r1 * (self.best_position - self.position)
        social_velocity = social * r2 * (global_best_position - self.position)
        self.velocity = inertia * self.velocity + cognitive_velocity + social_velocity

    def update_position(self, min_x, max_x):
        self.position += self.velocity
        self.position = max(min(self.position, max_x), min_x)
        fitness = fitness_function(self.position)
        if fitness > self.best_fitness:
            self.best_position = self.position
            self.best_fitness = fitness

def particle_swarm_optimization(pop_size, min_x, max_x, inertia, cognitive, social, iterations):
    swarm = [Particle(min_x, max_x) for _ in range(pop_size)]
    global_best_position = max(swarm, key=lambda p: p.best_fitness).best_position

    for iteration in range(iterations):
        for particle in swarm:
            particle.update_velocity(global_best_position, inertia, cognitive, social)
            particle.update_position(min_x, max_x)
            if fitness_function(particle.position) > fitness_function(global_best_position):
                global_best_position = particle.position

        print(f"Iteration {iteration + 1}: Global best = {global_best_position}, Fitness = {fitness_function(global_best_position)}")

    return global_best_position

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")

population_size = 30
min_value = -10
max_value = 10
inertia_weight = 0.5
cognitive_constant = 1.5
social_constant = 1.5
num_iterations = 5

best_solution = particle_swarm_optimization(population_size, min_value, max_value, inertia_weight, cognitive_constant, social_constant,
print(f"Best solution found: {best_solution}, Fitness: {fitness_function(best_solution)}")
```

```
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012
Iteration 1: Global best = 0.05841002841528997, Fitness = -0.003411731419474982
Iteration 2: Global best = 0.051530921649303674, Fitness = -0.002655435886026674
Iteration 3: Global best = 0.051530921649303674, Fitness = -0.002655435886026674
Iteration 4: Global best = 0.043711380087224416, Fitness = -0.0019106847491297992
Iteration 5: Global best = 0.043711380087224416, Fitness = -0.0019106847491297992
Best solution found: 0.043711380087224416, Fitness: -0.0019106847491297992
```

```
import numpy as np
import random

def create_distance_matrix(n_cities):
    np.random.seed(0)
    matrix = np.random.randint(1, 100, size=(n_cities, n_cities))
    np.fill_diagonal(matrix, 0)
    return matrix

n_cities = 10
n_ants = 20
n_iterations = 5
alpha = 1
beta = 2
evaporation_rate = 0.5
initial_pheromone = 1

distance_matrix = create_distance_matrix(n_cities)
pheromone_matrix = np.ones((n_cities, n_cities)) * initial_pheromone

class Ant:
    def __init__(self, n_cities):
        self.n_cities = n_cities
        self.route = []
        self.distance_travelled = 0

    def select_next_city(self, current_city, visited):
        probabilities = []
        for city in range(self.n_cities):
            if city not in visited:
                pheromone = pheromone_matrix[current_city][city] ** alpha
                heuristic = (1 / distance_matrix[current_city][city]) ** beta
                probabilities.append(pheromone * heuristic)
            else:
                probabilities.append(0)
        probabilities = np.array(probabilities) / sum(probabilities)
        next_city = np.random.choice(range(self.n_cities), p=probabilities)
        return next_city

    def find_route(self):
        current_city = random.randint(0, self.n_cities - 1)
        self.route = [current_city]
        visited = set(self.route)
        while len(visited) < self.n_cities:
            next_city = self.select_next_city(current_city, visited)
            self.route.append(next_city)
            self.distance_travelled += distance_matrix[current_city][next_city]
            visited.add(next_city)
            current_city = next_city
        self.distance_travelled += distance_matrix[self.route[-1]][self.route[0]]
        self.route.append(self.route[0])

def update_pheromones(ants):
    global pheromone_matrix
    pheromone_matrix *= (1 - evaporation_rate)
    for ant in ants:
        for i in range(len(ant.route) - 1):
            city_from = ant.route[i]
            city_to = ant.route[i + 1]
            pheromone_matrix[city_from][city_to] += 1.0 / ant.distance_travelled
            pheromone_matrix[city_to][city_from] += 1.0 / ant.distance_travelled

def ant_colony_optimization():
    best_route = None
    best_distance = float('inf')
    for iteration in range(n_iterations):
        ants = [Ant(n_cities) for _ in range(n_ants)]
        for ant in ants:
            ant.find_route()
            if ant.distance_travelled < best_distance:
                best_distance = ant.distance_travelled
                best_route = ant.route
        update_pheromones(ants)
        print(f"Iteration {iteration + 1}: Best distance = {best_distance}")
    return best_route, best_distance

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")

best_route, best_distance = ant_colony_optimization()
print(f"Best route found: {best_route} with distance: {best_distance}")
```

```
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012
Iteration 1: Best distance = 198
Iteration 2: Best distance = 139
Iteration 3: Best distance = 139
Iteration 4: Best distance = 139
Iteration 5: Best distance = 139
Best route found: [5, 4, 2, 6, 1, 3, 8, 9, 0, 7, 5] with distance: 139
```

```
import numpy as np
from scipy.special import gamma

def objective_function(x):
    return np.sum(x**2)

def levy_flight(alpha=1.5, size=1):
    sigma_u = np.power((gamma(1 + alpha) * np.sin(np.pi * alpha / 2) / gamma((1 + alpha) / 2)
                        * alpha * 2 ** ((alpha - 1) / 2)), 1 / alpha)

    u = np.random.normal(0, sigma_u, size)
    v = np.random.normal(0, 1, size)
    step = u / np.power(np.abs(v), 1 / alpha)
    return step

def cuckoo_search(objective_function, n_nests=25, max_iter=1000, pa=0.25):
    nests = np.random.uniform(low=-5, high=5, size=(n_nests, 2))
    fitness = np.apply_along_axis(objective_function, 1, nests)

    best_nest = nests[np.argmin(fitness)]
    best_fitness = np.min(fitness)

    for iteration in range(max_iter):
        for i in range(n_nests):
            new_nest = nests[i] + levy_flight(size=2)
            new_fitness = objective_function(new_nest)

            if new_fitness < fitness[i]:
                nests[i] = new_nest
                fitness[i] = new_fitness

        abandon = np.random.rand(n_nests) < pa
        nests[abandon] = np.random.uniform(low=-5, high=5, size=(np.sum(abandon), 2))

        current_best_nest = nests[np.argmin(fitness)]
        current_best_fitness = np.min(fitness)

        if current_best_fitness < best_fitness:
            best_nest = current_best_nest
            best_fitness = current_best_fitness

        print(f"Iteration {iteration + 1}, Best Fitness: {best_fitness}")

    return best_nest, best_fitness

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")

n_nests = 25
max_iter = 5
pa = 0.25

best_solution, best_value = cuckoo_search(objective_function, n_nests, max_iter, pa)

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")
print(f"\nBest solution: {best_solution}")
print(f"Best fitness value: {best_value}")
```

```
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012
Iteration 1, Best Fitness: 0.15085428329550082
Iteration 2, Best Fitness: 0.11619035727759061
Iteration 3, Best Fitness: 0.11619035727759061
Iteration 4, Best Fitness: 0.11619035727759061
Iteration 5, Best Fitness: 0.11619035727759061
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012

Best solution: [-0.2767273   0.19902854]
Best fitness value: 0.11619035727759061
```

```
import numpy as np

def obj_fn(x):
    return np.sum(x**2)

def gwo(obj_fn, dim, wolves, iters, lb, ub):
    pos = np.random.uniform(low=lb, high=ub, size=(wolves, dim))
    a_pos, b_pos, d_pos = np.zeros(dim), np.zeros(dim), np.zeros(dim)
    a_score, b_score, d_score = float("inf"), float("inf"), float("inf")

    for t in range(iters):
        for i in range(wolves):
            fit = obj_fn(pos[i])
            if fit < a_score:
                d_score, d_pos = b_score, b_pos.copy()
                b_score, b_pos = a_score, a_pos.copy()
                a_score, a_pos = fit, pos[i].copy()
            elif fit < b_score:
                d_score, d_pos = b_score, b_pos.copy()
                b_score, b_pos = fit, pos[i].copy()
            elif fit < d_score:
                d_score, d_pos = fit, pos[i].copy()

        a = 2 - t * (2 / iters)
        for i in range(wolves):
            for j in range(dim):
                r1, r2 = np.random.rand(), np.random.rand()
                A1, C1 = 2 * a * r1 - a, 2 * r2
                D_a = abs(C1 * a_pos[j] - pos[i, j])
                X1 = a_pos[j] - A1 * D_a

                r1, r2 = np.random.rand(), np.random.rand()
                A2, C2 = 2 * a * r1 - a, 2 * r2
                D_b = abs(C2 * b_pos[j] - pos[i, j])
                X2 = b_pos[j] - A2 * D_b

                r1, r2 = np.random.rand(), np.random.rand()
                A3, C3 = 2 * a * r1 - a, 2 * r2
                D_d = abs(C3 * d_pos[j] - pos[i, j])
                X3 = d_pos[j] - A3 * D_d

                pos[i, j] = (X1 + X2 + X3) / 3

            pos[i] = np.clip(pos[i], lb, ub)

        print(f"Iter {t+1}/{iters}, Best Score: {a_score}, Best Pos: {a_pos}")

    return a_score, a_pos

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")

dim = 5
wolves = 20
iters = 5
lb = -10
ub = 10

best_score, best_pos = gwo(obj_fn, dim, wolves, iters, lb, ub)

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")
print("\nFinal Best Score:", best_score)
print("Final Best Pos:", best_pos)
```

```
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012
Iter 1/5, Best Score: 21.715029777682197, Best Pos: [-1.38033578 -3.9686685   1.61670742 -0.93603481 -0.75463183]
Iter 2/5, Best Score: 5.4149339873086095, Best Pos: [ 0.6105372  -0.18940724  0.20313359 -0.97242011 -2.00485388]
Iter 3/5, Best Score: 2.2847774567123653, Best Pos: [-0.835901   -0.65080636  0.3569295  -1.00511556  0.15761373]
Iter 4/5, Best Score: 1.0758872919766413, Best Pos: [-0.21299378  0.5400246   0.0108156  -0.84788401  0.14096136]
Iter 5/5, Best Score: 0.7641923957535189, Best Pos: [-0.35708395  0.14864965 -0.00869101 -0.77937587 -0.08416916]
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012

Final Best Score: 0.7641923957535189
Final Best Pos: [-0.35708395  0.14864965 -0.00869101 -0.77937587 -0.08416916]
```

```
import numpy as np

def obj_fn(x):
    return np.sum(x**2)

def update_cell_state(cell, neighbors, lb, ub):
    new_state = np.mean(neighbors, axis=0)
    new_state = np.clip(new_state, lb, ub)
    return new_state

def parallel_cellular_algorithm(obj_fn, dim, grid_size, iterations, lb, ub):
    grid = np.random.uniform(lb, ub, (grid_size, grid_size, dim))
    best_solution = None
    best_fitness = float('inf')

    neighborhood = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for t in range(iterations):
        for i in range(grid_size):
            for j in range(grid_size):
                fitness = obj_fn(grid[i, j])

                if fitness < best_fitness:
                    best_fitness = fitness
                    best_solution = grid[i, j]

        new_grid = np.copy(grid)
        for i in range(grid_size):
            for j in range(grid_size):
                neighbors = []
                for dx, dy in neighborhood:
                    ni, nj = (i + dx) % grid_size, (j + dy) % grid_size
                    neighbors.append(grid[ni, nj])

                new_grid[i, j] = update_cell_state(grid[i, j], neighbors, lb, ub)

        grid = np.copy(new_grid)

        print(f"Iteration {t+1}/{iterations}, Best Fitness: {best_fitness}")

    return best_fitness, best_solution

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")

dim = 5
grid_size = 10
iterations = 5
lb = -10
ub = 10

best_fitness, best_solution = parallel_cellular_algorithm(obj_fn, dim, grid_size, iterations, lb, ub)

print("StudentName: Adikar Charvi Sree Teja")
print("USN: 1BM22CS012")
print("\nFinal Best Fitness:", best_fitness)
print("Final Best Solution:", best_solution)
```

```
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012
Iteration 1/5, Best Fitness: 25.2506917295739
Iteration 2/5, Best Fitness: 1.778196721479161
Iteration 3/5, Best Fitness: 1.778196721479161
Iteration 4/5, Best Fitness: 1.2298236083547542
Iteration 5/5, Best Fitness: 0.5538730603681196
StudentName: Adikar Charvi Sree Teja
USN: 1BM22CS012

Final Best Fitness: 0.5538730603681196
Final Best Solution: [-0.12146374 -0.00652971  0.68535678  0.20920674  0.15998626]
```

```
import numpy as np

def objective_function(x):
    return np.sum(x**2)

def gene_expression_algorithm(objective_function, pop_size=50, num_genes=5, mutation_rate=0.1, crossover_rate=0.7, num_generations=5):
    population = np.random.uniform(-5, 5, (pop_size, num_genes))
    fitness = np.apply_along_axis(objective_function, 1, population)

    best_solution = population[np.argmin(fitness)]
    best_fitness = np.min(fitness)

    for generation in range(num_generations):
        new_population = []

        for _ in range(pop_size):
            parent1, parent2 = selection(population, fitness)
            offspring = crossover(parent1, parent2, crossover_rate)
            offspring = mutation(offspring, mutation_rate)
            new_population.append(offspring)

        population = np.array(new_population)
        fitness = np.apply_along_axis(objective_function, 1, population)

        best_idx = np.argmin(fitness)
        current_best_solution = population[best_idx]
        current_best_fitness = fitness[best_idx]

        if current_best_fitness < best_fitness:
            best_solution = current_best_solution
            best_fitness = current_best_fitness

        print(f"Generation {generation + 1}, Best Fitness: {best_fitness}")

    print("StudentName: Adikar Charvi Sree Teja USN:1BM22CS012")
    return best_solution, best_fitness

def selection(population, fitness):
    idx = np.random.choice(len(population), 2, replace=False)
    return population[idx[0]], population[idx[1]]

def crossover(parent1, parent2, crossover_rate):
    if np.random.rand() < crossover_rate:
        mask = np.random.rand(len(parent1)) > 0.5
        offspring = np.where(mask, parent1, parent2)
        return offspring
    else:
        return parent1.copy()

def mutation(offspring, mutation_rate):
    for i in range(len(offspring)):
        if np.random.rand() < mutation_rate:
            offspring[i] += np.random.uniform(-0.5, 0.5)
    return offspring

pop_size = 50
num_genes = 5
mutation_rate = 0.1
crossover_rate = 0.7
num_generations = 5

best_solution, best_fitness = gene_expression_algorithm(objective_function, pop_size, num_genes, mutation_rate, crossover_rate, num_gener

print(f"\nBest Solution: {best_solution}")
print(f"Best Fitness: {best_fitness}")
```

```
Generation 1, Best Fitness: 9.35170269951083
Generation 2, Best Fitness: 9.35170269951083
Generation 3, Best Fitness: 9.35170269951083
Generation 4, Best Fitness: 9.35170269951083
Generation 5, Best Fitness: 6.873192749650595
StudentName: Adikar Charvi Sree Teja USN:1BM22CS012

Best Solution: [ 0.92457583  2.34535539  0.06184925 -0.568565   -0.43654198]
Best Fitness: 6.873192749650595
```