```python
import numpy as np

# Objective function: Minimize the weight of the truss
def fitness(x):
    # Truss example: A simple 2D truss with 3 members and 2 nodes
    # Here x = [A1, A2, A3] are the cross-sectional areas of the members
    A1, A2, A3 = x

    # Define lengths of members (in meters) and material properties
    L1 = 10  # Length of member 1 (m)
    L2 = 10  # Length of member 2 (m)
    L3 = 10  # Length of member 3 (m)
    E = 200e9  # Young's Modulus (Pa) (for steel)
    rho = 7800  # Density of steel (kg/m^3)

    # Calculate the weight of the truss (total mass * gravity)
    mass = (rho * A1 * L1 + rho * A2 * L2 + rho * A3 * L3)  # in kg
    weight = mass * 9.81  # in Newtons

    return weight

# Constraint functions: Stress and deflection
# For simplicity, assume simple linear elastic behavior and ideal conditions

def stress_constraint(x):
    # Example stress limit (simplified)
    sigma_max = 250e6  # Maximum allowable stress (Pa) for steel
    A1, A2, A3 = x
    F1 = 10000  # Load on member 1 (N)
    F2 = 10000  # Load on member 2 (N)
    F3 = 10000  # Load on member 3 (N)

    stress_1 = F1 / A1  # Stress in member 1
    stress_2 = F2 / A2  # Stress in member 2
    stress_3 = F3 / A3  # Stress in member 3

    # Return violation if any member exceeds allowable stress
    return max(0, stress_1 - sigma_max), max(0, stress_2 - sigma_max), max(0, stress_3 - sigma_max)

def deflection_constraint(x):
    # Example deflection limit (simplified)
    delta_max = 0.01  # Maximum allowable deflection (m)
    A1, A2, A3 = x
    F1 = 10000  # Load on member 1 (N)
    F2 = 10000  # Load on member 2 (N)
    F3 = 10000  # Load on member 3 (N)
    L1 = 10  # Length of member 1 (m)
    L2 = 10  # Length of member 2 (m)
    L3 = 10  # Length of member 3 (m)
    E = 200e9  # Young's Modulus (Pa)

    # Simplified deflection calculation (linear approximation)
    delta_1 = (F1 * L1**3) / (3 * E * A1)  # Deflection in member 1
    delta_2 = (F2 * L2**3) / (3 * E * A2)  # Deflection in member 2
    delta_3 = (F3 * L3**3) / (3 * E * A3)  # Deflection in member 3

    # Return violation if any member exceeds allowable deflection
    return max(0, delta_1 - delta_max), max(0, delta_2 - delta_max), max(0, delta_3 - delta_max)

# Grey Wolf Optimizer class
class GreyWolfOptimizer:
    def __init__(self, fitness_func, constraints, dim, lower_bound, upper_bound, num_wolves=30, max_iter=100):
        self.fitness_func = fitness_func  # Fitness function
        self.constraints = constraints  # List of constraint functions
        self.dim = dim  # Number of dimensions (variables to optimize)
        self.lower_bound = lower_bound  # Lower bound for the design variables
        self.upper_bound = upper_bound  # Upper bound for the design variables
        self.num_wolves = num_wolves  # Number of wolves
```

```
        self.num_wolves = num_wolves  # Number of wolves
        self.max_iter = max_iter  # Maximum number of iterations

        # Initialize positions of wolves
        self.wolves = np.random.uniform(low=self.lower_bound, high=self.upper_bound, size=(self.num_wolves, self.c
        self.alpha_pos = np.zeros(self.dim)  # Position of alpha wolf
        self.beta_pos = np.zeros(self.dim)  # Position of beta wolf
        self.delta_pos = np.zeros(self.dim)  # Position of delta wolf
        self.alpha_score = float("inf")  # Fitness of alpha wolf
        self.beta_score = float("inf")  # Fitness of beta wolf
        self.delta_score = float("inf")  # Fitness of delta wolf

    def update_positions(self, a):
        # Update the positions of wolves based on alpha, beta, delta
        for i in range(self.num_wolves):
            A1 = 2 * a * np.random.rand(self.dim) - a
            C1 = 2 * np.random.rand(self.dim)
            D_alpha = np.abs(C1 * self.alpha_pos - self.wolves[i])
            X1 = self.alpha_pos - A1 * D_alpha

            A2 = 2 * a * np.random.rand(self.dim) - a
            C2 = 2 * np.random.rand(self.dim)
            D_beta = np.abs(C2 * self.beta_pos - self.wolves[i])
            X2 = self.beta_pos - A2 * D_beta

            A3 = 2 * a * np.random.rand(self.dim) - a
            C3 = 2 * np.random.rand(self.dim)
            D_delta = np.abs(C3 * self.delta_pos - self.wolves[i])
            X3 = self.delta_pos - A3 * D_delta

            # Update wolf's position
            self.wolves[i] = (X1 + X2 + X3) / 3
            # Ensure wolves stay within the search space
            self.wolves[i] = np.clip(self.wolves[i], self.lower_bound, self.upper_bound)

    def optimize(self):
        for t in range(self.max_iter):
            a = 2 - t * (2 / self.max_iter)

            # Evaluate fitness for each wolf
            for i in range(self.num_wolves):
                fitness_value = self.fitness_func(self.wolves[i])

                # Check for constraint violations (penalty approach)
                stress_penalty = sum(self.constraints[0](self.wolves[i]))
                deflection_penalty = sum(self.constraints[1](self.wolves[i]))
                total_penalty = stress_penalty + deflection_penalty

                # Total fitness value with penalties
                fitness_value += total_penalty

                # Update alpha, beta, and delta wolves
                if fitness_value < self.alpha_score:
                    self.alpha_score = fitness_value
                    self.alpha_pos = self.wolves[i].copy()
                elif fitness_value < self.beta_score:
                    self.beta_score = fitness_value
                    self.beta_pos = self.wolves[i].copy()
                elif fitness_value < self.delta_score:
                    self.delta_score = fitness_value
                    self.delta_pos = self.wolves[i].copy()

            # Update wolves' positions
            self.update_positions(a)

            # Print the progress
            print(f"Iteration {t+1}/{self.max_iter}, Best fitness = {self.alpha_score}")

        return self.alpha_pos, self.alpha_score
```

```
# Parameters for the optimization problem
dim = 3  # Three members in the truss
lower_bound = 1e-4  # Lower bound for the cross-sectional areas
upper_bound = 0.5  # Upper bound for the cross-sectional areas
num_wolves = 30  # Number of wolves
max_iter = 100  # Maximum number of iterations

# Initialize and run the Grey Wolf Optimizer
gwo = GreyWolfOptimizer(fitness_func=fitness, constraints=[stress_constraint, deflection_constraint], dim=dim, lov
best_position, best_fitness = gwo.optimize()

# Output the result
print(f"Optimal Cross-sectional Areas: {best_position}")
print(f"Minimum Weight: {best_fitness} N")
```

```
Iteration 1/100, Best fitness = 124608.13687813526
Iteration 2/100, Best fitness = 72804.98567007676
Iteration 3/100, Best fitness = 64316.09064478929
Iteration 4/100, Best fitness = 1914.2302890130575
Iteration 5/100, Best fitness = 230.02400000000003
Iteration 6/100, Best fitness = 230.02400000000003
Iteration 7/100, Best fitness = 230.02400000000003
Iteration 8/100, Best fitness = 230.02400000000003
Iteration 9/100, Best fitness = 230.02400000000003
Iteration 10/100, Best fitness = 230.02400000000003
Iteration 11/100, Best fitness = 230.02400000000003
Iteration 12/100, Best fitness = 230.02400000000003
Iteration 13/100, Best fitness = 230.02400000000003
Iteration 14/100, Best fitness = 230.02400000000003
Iteration 15/100, Best fitness = 230.02400000000003
Iteration 16/100, Best fitness = 230.02400000000003
Iteration 17/100, Best fitness = 230.02400000000003
Iteration 18/100, Best fitness = 230.02400000000003
Iteration 19/100, Best fitness = 230.02400000000003
Iteration 20/100, Best fitness = 230.02400000000003
Iteration 21/100, Best fitness = 230.02400000000003
Iteration 22/100, Best fitness = 230.02400000000003
Iteration 23/100, Best fitness = 230.02400000000003
Iteration 24/100, Best fitness = 230.02400000000003
Iteration 25/100, Best fitness = 230.02400000000003
Iteration 26/100, Best fitness = 230.02400000000003
Iteration 27/100, Best fitness = 230.02400000000003
Iteration 28/100, Best fitness = 230.02400000000003
Iteration 29/100, Best fitness = 230.02400000000003
Iteration 30/100, Best fitness = 230.02400000000003
Iteration 31/100, Best fitness = 230.02400000000003
Iteration 32/100, Best fitness = 230.02400000000003
Iteration 33/100, Best fitness = 230.02400000000003
Iteration 34/100, Best fitness = 230.02400000000003
Iteration 35/100, Best fitness = 230.02400000000003
Iteration 36/100, Best fitness = 230.02400000000003
Iteration 37/100, Best fitness = 230.02400000000003
Iteration 38/100, Best fitness = 230.02400000000003
Iteration 39/100, Best fitness = 230.02400000000003
Iteration 40/100, Best fitness = 230.02400000000003
Iteration 41/100, Best fitness = 230.02400000000003
Iteration 42/100, Best fitness = 230.02400000000003
Iteration 43/100, Best fitness = 230.02400000000003
Iteration 44/100, Best fitness = 230.02400000000003
Iteration 45/100, Best fitness = 230.02400000000003
Iteration 46/100, Best fitness = 230.02400000000003
Iteration 47/100, Best fitness = 230.02400000000003
Iteration 48/100, Best fitness = 230.02400000000003
Iteration 49/100, Best fitness = 230.02400000000003
Iteration 50/100, Best fitness = 230.02400000000003
Iteration 51/100, Best fitness = 230.02400000000003
Iteration 52/100, Best fitness = 230.02400000000003
Iteration 53/100, Best fitness = 230.02400000000003
Iteration 54/100, Best fitness = 230.02400000000003
Iteration 55/100, Best fitness = 230.02400000000003
Iteration 56/100, Best fitness = 230.02400000000003
Iteration 57/100, Best fitness = 230.02400000000003
```