```
import numpy as np
import random

# Define the problem: Traffic control optimization
def congestion_function(signal_timings):
    """
    Simulated function to measure traffic congestion.
    Lower values indicate better traffic flow.
    """
    return np.sum((signal_timings - ideal_timings)**2)

# Parameters for the problem
num_signals = 4  # Number of traffic signals
ideal_timings = np.array([30, 40, 50, 60])  # Ideal timings for minimal congestion

# Cuckoo Search Parameters
num_nests = 10  # Number of nests (solutions)
num_iterations = 100  # Number of iterations
pa = 0.25  # Discovery probability
bounds = [(10, 90)] * num_signals  # Timing bounds for each signal

# Lévy flight function
def levy_flight(Lambda=1.5):
    sigma = (np.math.gamma(1 + Lambda) * np.sin(np.pi * Lambda / 2) /
             (np.math.gamma((1 + Lambda) / 2) * Lambda * 2**((Lambda - 1) /
2)))**(1 / Lambda)
    u = np.random.normal(0, sigma, size=num_signals)
    v = np.random.normal(0, 1, size=num_signals)
    step = u / abs(v)**(1 / Lambda)
    return step

# Initialize nests (random solutions)
def initialize_nests(num_nests, bounds):
    return np.array([[random.uniform(low, high) for low, high in bounds] for _ in
range(num_nests)])

# Replace worst nests
def replace_worst_nests(nests, fitness, pa, bounds):
    num_replacements = int(pa * len(nests))
    worst_indices = np.argsort(fitness)[-num_replacements:]
    for idx in worst_indices:
        nests[idx] = np.array([random.uniform(low, high) for low, high in bounds])
    return nests

# Cuckoo Search Algorithm
def cuckoo_search():
    nests = initialize_nests(num_nests, bounds)
    best_nest = None
    best_fitness = float('inf')
```

```python
    for iteration in range(num_iterations):
        # Fitness evaluation
        fitness = np.array([congestion_function(nest) for nest in nests])

        # Find the best nest
        if np.min(fitness) < best_fitness:
            best_fitness = np.min(fitness)
            best_nest = nests[np.argmin(fitness)]

        # Generate new solutions via Lévy flights
        new_nests = np.array([nest + levy_flight() for nest in nests])
        new_nests = np.clip(new_nests, [low for low, high in bounds], [high for
low, high in bounds])

        # Evaluate new fitness
        new_fitness = np.array([congestion_function(nest) for nest in new_nests])

        # Select better solutions
        for i in range(num_nests):
            if new_fitness[i] < fitness[i]:
                nests[i] = new_nests[i]

        # Abandon worst nests
        nests = replace_worst_nests(nests, fitness, pa, bounds)

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

    return best_nest, best_fitness

# Run the Cuckoo Search algorithm
best_solution, best_fitness = cuckoo_search()

# Output the results
print("\nOptimal Signal Timings:", best_solution)
print("Minimal Congestion Measure:", best_fitness)
```