CODE:

```
import numpy as np
import random

# Lévy flight step
def levy_flight(Lambda):
    u = np.random.normal(0, 1, 1)
    v = np.random.normal(0, 1, 1)
    step = u / abs(v)**(1 / Lambda)
    return step[0]

# Traffic congestion function (Objective Function)
def traffic_objective(signal_timings):
    """
    Simulates traffic congestion based on signal timings.
    Lower values represent better performance (less congestion).
    Example: The function is simplified and can be replaced with real traffic data.
    """
    # Example congestion model (you can replace this with a real simulation):
    total_waiting_time = 0
    for timing in signal_timings:
        total_waiting_time += abs(50 - timing) + random.uniform(0, 5)
    return total_waiting_time

class CuckooSearch:
    def __init__(self, num_nests, num_iterations, discovery_rate, bounds):
        self.num_nests = num_nests
        self.num_iterations = num_iterations
        self.discovery_rate = discovery_rate
        self.bounds = bounds
        self.dimensions = len(bounds)  # Number of traffic signals
        self.nests = np.random.uniform(
            [b[0] for b in bounds], [b[1] for b in bounds], (num_nests, self.dimensions)
        )
        self.best_nest = self.nests[0]
        self.best_fitness = float("inf")
```

```
    def run(self):
        for iteration in range(self.num_iterations):
            new_nests = self.generate_new_solutions()
            self.evaluate_nests(new_nests)
            self.abandon_worst_nests()
            print(f"Iteration {iteration + 1}/{self.num_iterations}, Best Fitness: 
{self.best_fitness}")
        return self.best_nest, self.best_fitness

    def evaluate_nests(self, new_nests):
        for i in range(len(new_nests)):
            fitness = traffic_objective(new_nests[i])
            if fitness < self.best_fitness:
                self.best_fitness = fitness
                self.best_nest = new_nests[i]

    def generate_new_solutions(self):
        new_nests = []
        for nest in self.nests:
            step = levy_flight(Lambda=1.5)
            new_solution = nest + step * np.random.uniform(-1, 1, self.dimensions)
            new_solution = np.clip(new_solution, [b[0] for b in self.bounds], [b[1] for b in 
self.bounds])
            new_nests.append(new_solution)
        return np.array(new_nests)

    def abandon_worst_nests(self):
        discovery_mask = np.random.rand(self.num_nests) < self.discovery_rate
        for i in range(self.num_nests):
            if discovery_mask[i]:
                self.nests[i] = np.random.uniform(
                    [b[0] for b in self.bounds], [b[1] for b in self.bounds]
                )

# Parameters
num_signals = 5  # Number of traffic signals
bounds = [(20, 100)] * num_signals  # Traffic light timings (min, max) in seconds
```

```
num_nests = 20
num_iterations = 50
discovery_rate = 0.25

# Run Cuckoo Search
cs = CuckooSearch(num_nests, num_iterations, discovery_rate, bounds)
best_solution, best_fitness = cs.run()

# Output results
print("\nOptimal Traffic Signal Timings:")
print(f"Best Signal Timings: {best_solution}")
print(f"Best Fitness (Minimized Congestion): {best_fitness:.2f}")
```

OUTPUT:

```
Iteration 1/5, Best Fitness: 72.28102682019099
Iteration 2/5, Best Fitness: 63.84313266058888
Iteration 3/5, Best Fitness: 63.84313266058888
Iteration 4/5, Best Fitness: 63.84313266058888
Iteration 5/5, Best Fitness: 63.84313266058888

Optimal Traffic Signal Timings:
Best Signal Timings: [49.2411359  37.16669285 38.6408849  46.76216812
    27.92477078]
Best Fitness (Minimized Congestion): 63.84
```