```python
import numpy as np
import random
from scipy.special import gamma  # Import the gamma function

# Objective function (fitness function) to minimize waiting time at the intersection
def traffic_waiting_time(signal_timings):
    # This is a simplified model of the traffic flow; the function can be more complex.
    flow = np.array([1000, 1200, 1100, 1300])  # hypothetical vehicle flow for each phase (vehicles per hour)
    max_capacity = np.array([1500, 1500, 1500, 1500])  # maximum capacity for each phase (vehicles per hour)

    # The waiting time can be roughly modeled as the inverse of the flow divided by the capacity.
    waiting_time = np.sum((signal_timings * flow) / max_capacity)  # total waiting time for vehicles
    return waiting_time

# Levy flight for exploration
def levy_flight(dim, beta=1.5):
    sigma_u = (gamma(1 + beta) * np.sin(np.pi * beta / 2) /
               (gamma((1 + beta) / 2) * beta * 2 ** ((beta - 1) / 2))) ** (1 / beta)
    u = np.random.normal(0, sigma_u, dim)
    v = np.random.normal(0, 1, dim)
    step = u / (np.abs(v) ** (1 / beta))
    return step

# Cuckoo Search algorithm
def cuckoo_search(num_nests=50, num_iterations=100, dim=4, alpha=0.01, beta=1.5, pa=0.25):
    # Initialize the nests with random values (signal timings)
    nests = np.random.uniform(10, 60, (num_nests, dim))  # Signal timings between 10 and 60 seconds for each phase
    fitness = np.array([traffic_waiting_time(nest) for nest in nests])  # Initial fitness for each nest

    # Find the best solution
    best_nest = nests[np.argmin(fitness)]
    best_fitness = np.min(fitness)

    # Main loop
    for iteration in range(num_iterations):
        # Generate a new solution using Levy flight
        new_nests = nests + alpha * levy_flight(dim, beta)  # Perturb the nests

        # Apply bounds to signal timings (e.g., between 10 and 60 seconds)
        new_nests = np.clip(new_nests, 10, 60)

        # Evaluate the fitness of the new nests
        new_fitness = np.array([traffic_waiting_time(nest) for nest in new_nests])

        # Replace worse nests with better ones
        for i in range(num_nests):
            if new_fitness[i] < fitness[i]:
                nests[i] = new_nests[i]
                fitness[i] = new_fitness[i]

        # Replace some nests randomly based on probability 'pa'
        for i in range(num_nests):
            if random.random() < pa:
                nests[i] = np.random.uniform(10, 60, dim)  # Randomly reinitialize the nest
                fitness[i] = traffic_waiting_time(nests[i])

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

        # Print progress for each iteration
        print(f"Iteration {iteration + 1}, Best waiting time: {best_fitness}")

    return best_nest, best_fitness

# Run Cuckoo Search for traffic signal optimization
best_signal_timings, best_waiting_time = cuckoo_search(num_nests=50, num_iterations=100, dim=4)

# Output the best signal timings and their corresponding waiting time
print("\nBest traffic signal timings (seconds per phase):", best_signal_timings)
print("Best waiting time:", best_waiting_time)
```

```
Iteration 52, Best waiting time: 43.31852262505121
Iteration 53, Best waiting time: 43.31852262505121
Iteration 54, Best waiting time: 43.31852262505121
Iteration 55, Best waiting time: 43.31852262505121
Iteration 56, Best waiting time: 43.31852262505121
Iteration 57, Best waiting time: 43.31852262505121
Iteration 58, Best waiting time: 43.31852262505121
Iteration 59, Best waiting time: 43.31852262505121
Iteration 60, Best waiting time: 43.31852262505121
Iteration 61, Best waiting time: 43.31852262505121
Iteration 62, Best waiting time: 43.31852262505121
Iteration 63, Best waiting time: 43.31852262505121
Iteration 64, Best waiting time: 43.31852262505121
Iteration 65, Best waiting time: 43.31852262505121
Iteration 66, Best waiting time: 43.31852262505121
Iteration 67, Best waiting time: 43.31852262505121
Iteration 68, Best waiting time: 43.31852262505121
Iteration 69, Best waiting time: 43.31852262505121
Iteration 70, Best waiting time: 43.31852262505121
Iteration 71, Best waiting time: 43.31852262505121
Iteration 72, Best waiting time: 43.31852262505121
Iteration 73, Best waiting time: 43.31852262505121
Iteration 74, Best waiting time: 43.31852262505121
Iteration 75, Best waiting time: 43.31852262505121
Iteration 76, Best waiting time: 43.31852262505121
Iteration 77, Best waiting time: 43.31852262505121
Iteration 78, Best waiting time: 43.31852262505121
Iteration 79, Best waiting time: 43.31852262505121
Iteration 80, Best waiting time: 43.31852262505121
Iteration 81, Best waiting time: 43.31852262505121
Iteration 82, Best waiting time: 43.31852262505121
Iteration 83, Best waiting time: 43.31852262505121
Iteration 84, Best waiting time: 43.31852262505121
Iteration 85, Best waiting time: 43.31852262505121
Iteration 86, Best waiting time: 43.31852262505121
Iteration 87, Best waiting time: 43.31852262505121
Iteration 88, Best waiting time: 43.31852262505121
Iteration 89, Best waiting time: 43.31852262505121
Iteration 90, Best waiting time: 43.31852262505121
Iteration 91, Best waiting time: 43.31852262505121
Iteration 92, Best waiting time: 43.31852262505121
Iteration 93, Best waiting time: 43.31852262505121
Iteration 94, Best waiting time: 43.31852262505121
Iteration 95, Best waiting time: 43.31852262505121
Iteration 96, Best waiting time: 43.31852262505121
Iteration 97, Best waiting time: 43.31852262505121
Iteration 98, Best waiting time: 43.31852262505121
Iteration 99, Best waiting time: 43.31852262505121
Iteration 100, Best waiting time: 43.31852262505121

Best traffic signal timings (seconds per phase): [33.85108406 41.19955268 10.56663762 51.28594153]
Best waiting time: 43.31852262505121
```