CAPSTONE PROJECT

CSA0612 – Design and Analysis of Algorithms for Optimization

"OPTIMIZING TRAFFIC FLOW IN A SMART CITY"

REPORT SUBMITTED BY:

Register Numbers	**192324198,192311340**
Names	**HARIGOWTHAM. A PRAVIN GANESH JAMBULINGAM**

GUIDED BY:

Dr. SURESH P

TITLE: OPTIMIZING TRAFFIC FLOW IN A SMART CITY

Problem Statement:

"Enhancing Traffic Management in a Smart City"
Traffic jams pose a significant challenge in expanding urban areas, resulting in prolonged delays, excessive fuel consumption, and increased pollution. Conventional traffic light systems fail to effectively manage changing traffic conditions.
This project's aim is to create a system that utilizes real-time information to modify traffic signal timings and enhance traffic movement.

Key objectives:

- Decrease waiting periods at intersections.
- Reduce total congestion levels.
- Facilitate the movement of emergency and priority vehicles.

The project will include simulating traffic scenarios, developing an algorithm for adaptive traffic signal control, and assessing improvements in average delays and the efficiency of traffic flow.

Introduction:

In contemporary urban settings, traffic congestion presents considerable challenges, such as prolonged travel durations, increased fuel usage, and heightened air pollution. Conventional fixed-timing traffic signal systems lack the adaptability needed to manage changing traffic conditions, resulting in inefficiencies and delays.

Smart city initiatives seek to address these problems by utilizing advanced technologies to enhance traffic management. By harnessing real-time data from IoT sensors, cameras, and connected vehicles, adaptive traffic signal systems can adjust signal timings dynamically to optimize the flow of vehicles at intersections.

- ✓ **Objectives:**
- ❖ Decrease Waiting Times: Reduce delays for vehicles at intersections by responding to current traffic conditions.
- ❖ Enhance Traffic Flow: Lessen overall congestion by facilitating smoother vehicle movement.
- ❖ Prioritize Emergency Vehicles: Allow emergency services and public transportation to pass through traffic effectively.
- ❖ Increase Scalability: Ensure the system can be expanded to accommodate more intersections as the city develops.
- ❖ Reduce Environmental Impact: Lower fuel consumption and emissions by optimizing traffic movement.
- ✓ **Key Features:**
- ❖ Real-Time Data Integration: Utilize sensors, cameras, and vehicle-to-infrastructure communication to gather traffic information.
- ❖ Adaptive Signal Control: Modify signal timings in real-time based on traffic trends.
- ❖ Traffic Prediction: Use AI/ML algorithms to foresee congestion and manage traffic proactively.
- ❖ Emergency and Priority Management: Identify and give precedence to emergency vehicles and public transportation.
- ❖ Performance Metrics: Assess and monitor system performance using crucial metrics such as average delays and emission levels.

This project intends to develop an efficient and sustainable traffic management solution that improves mobility and aligns with the goal of a smarter, more attractive urban environment.

<u>Literature Review</u>:

This literature review concentrates on assessing current research, methodologies, and technologies aimed at improving traffic flow in smart cities. The following are significant insights and strategies derived from prior studies:

1. Conventional Traffic Management Systems

- Fixed-Timing Signals: These systems operate with predetermined signal timings based on historical trends, which become ineffective under varying traffic circumstances.

- Limitations: The inability to adjust to immediate changes results in longer wait times and increased congestion.

2. Dynamic Traffic Signal Control Systems

- Overview: Adaptive systems modify signal timings in real-time based on current data.

- Example Approaches:

- SCATS (Sydney Coordinated Adaptive Traffic System): Modifies signal timings using information gathered from traffic detectors.

- SCOOT (Split Cycle Offset Optimization Technique): Utilizes traffic flow data to decrease delays and enhance cycle durations.

- Obstacles: High implementation costs and complex integration processes.

3. Artificial Intelligence (AI) in Traffic Management

- Machine Learning Models: Forecast traffic trends and make proactive signal adjustments.

- Deep Reinforcement Learning: Allows traffic systems to learn the best signal strategies through simulation environments.

- Applications: Research has indicated up to a 30% enhancement in traffic flow through AI-driven methods.

4. IoT and Sensor-Driven Systems

- IoT Integration: Devices such as sensors, cameras, and GPS units gather real-time data on traffic volume and movement.

- Vehicle-to-Infrastructure (V2I) Communication: Improves system responsiveness by linking vehicles directly to traffic signals.

- Benefits: Reliable data acquisition and quicker decision-making processes.

5. Optimization Methods

- Heuristic Algorithms: Techniques like Genetic Algorithms (GA) and Ant Colony Optimization (ACO) are implemented to refine signal timings.

- Multi-Objective Optimization: Balances various aspects such as reducing delays, maximizing throughput, and controlling emissions.

6. Simulation Tools for Traffic Studies

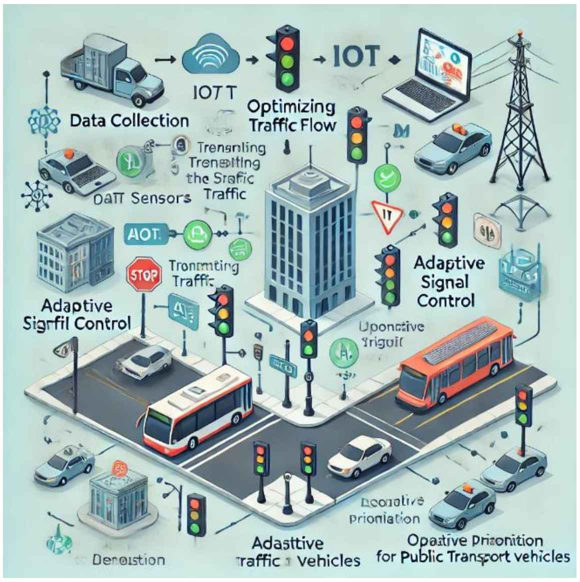
- SUMO (Simulation of Urban Mobility): Commonly employed for modeling and analyzing traffic situations.
- Cisco Packet Tracer: Valuable for simulating interactions between networked devices in smart urban environments.

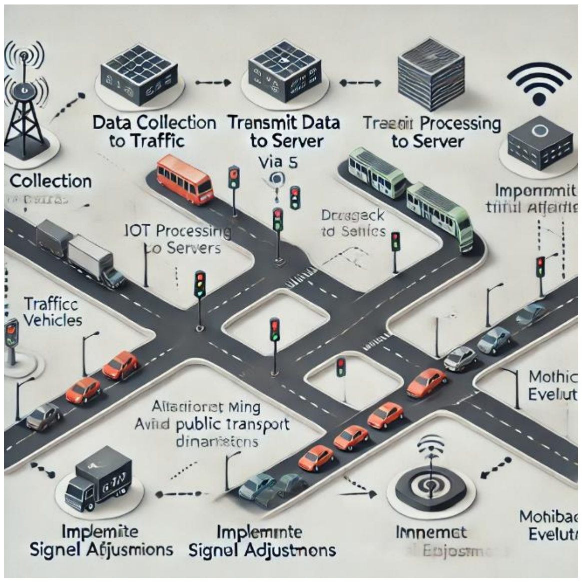
7. Challenges and Shortcomings

- Data Availability: Real-time data acquisition continues to be insufficient in many areas.
- Scalability: Numerous systems find it challenging to scale with the expanding size of cities.
- Cost-Effectiveness: Significant initial costs hinder broad implementation.

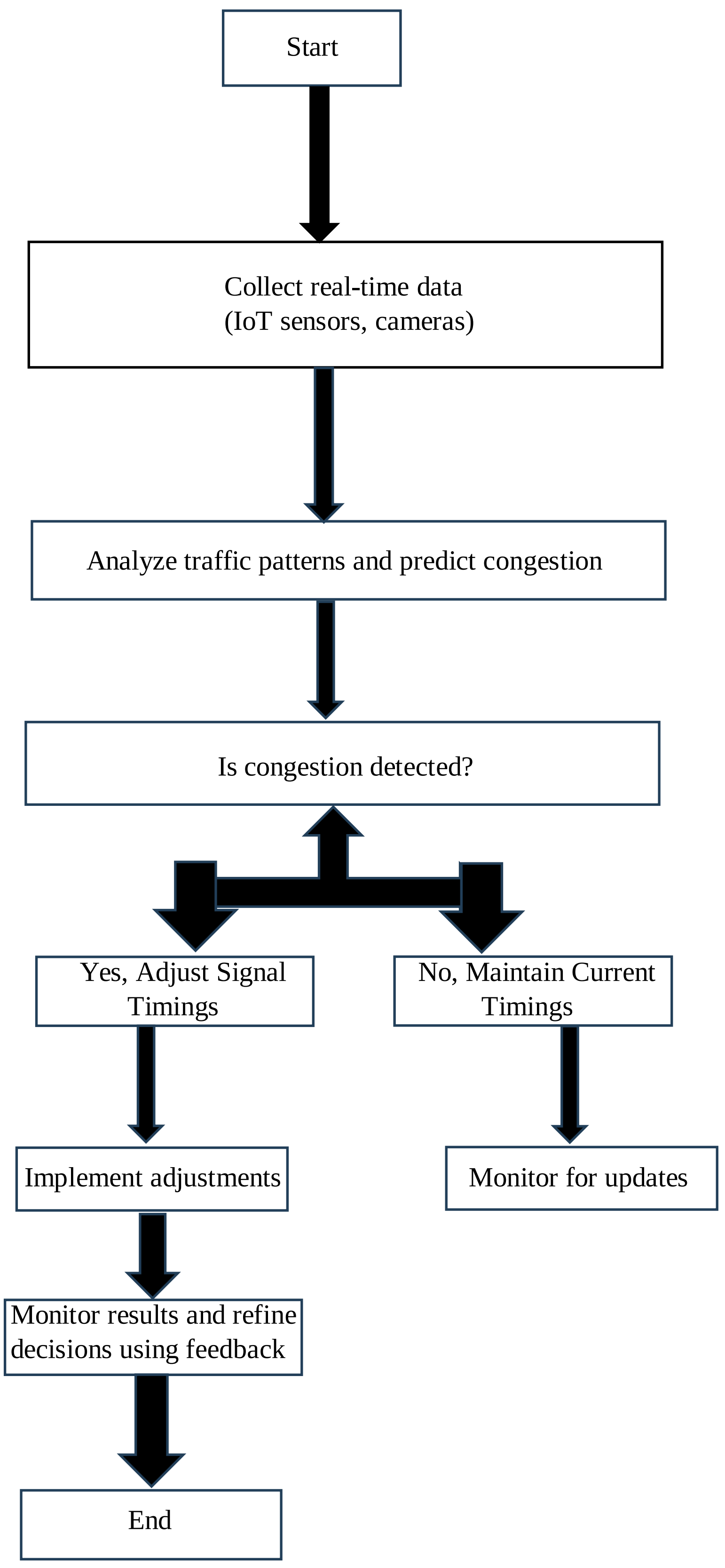
This literature review underscores the promise of adaptive signal systems, AI, and IoT in addressing urban traffic challenges. However, issues like costs, scalability, and data reliability must be resolved to facilitate effective implementation.

<u>Architecture Diagram:</u>




Data Collection to Traffic
Transmit Data to Server
Processing to Server
Collection
IOT Processing to Servers
Trafficc Vehicles
Implemite Signel Afjusmions
Implemnte Signel Adjustmons

Flowchart for problem solving:


Start
Collect real-time data
(IoT sensors, cameras)
Analyze traffic patterns and predict congestion
Is congestion detected?
Yes, Adjust Signal Timings
No, Maintain Current Timings
Implement adjustments
Monitor for updates
Monitor results and refine decisions using feedback
End

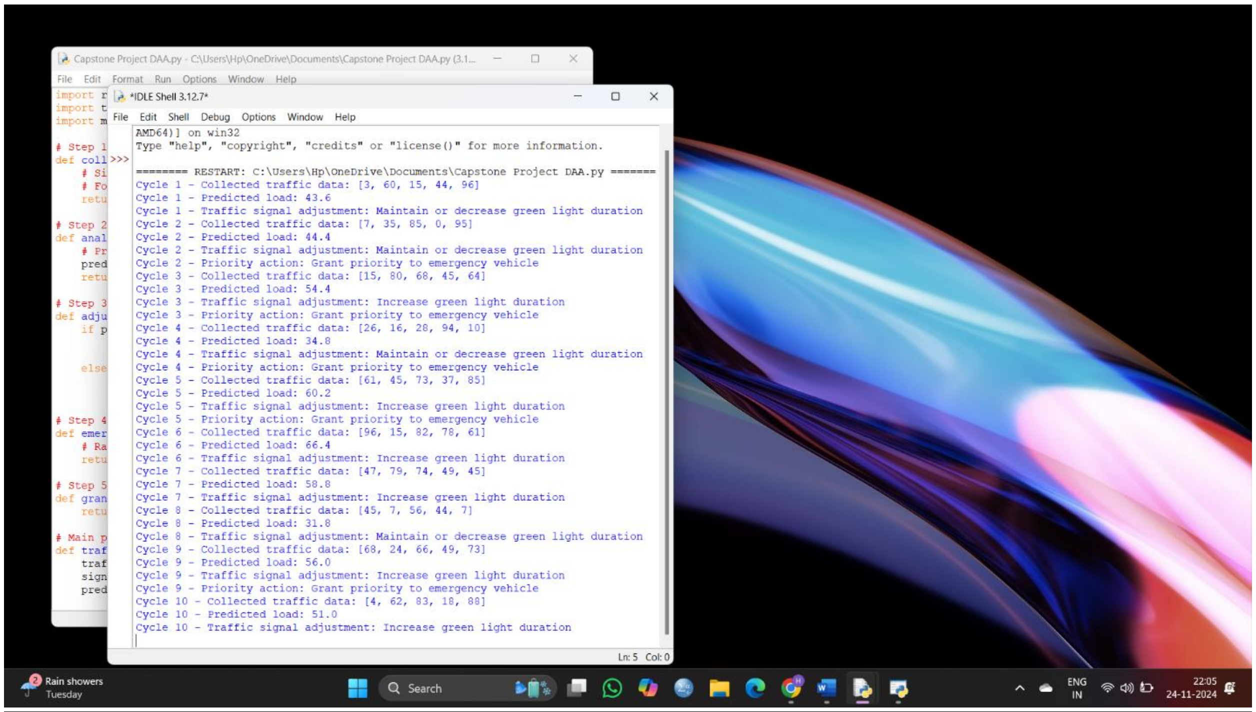
Pseudocode:

```
START
// Step 1: Data Collection
Collect real-time data from IoT sensors, cameras, and vehicle-to-infrastructure systems
TrafficData = collect_traffic_data()
// Step 2: Data Analysis
Analyze current traffic conditions and predict future load
PredictedLoad = analyze_traffic_patterns(TrafficData)
// Step 3: Decision Making
If PredictedLoad > average load then
    // Step 4: Adjust Traffic Signal Timings
    adjust_traffic_signals('increase', PredictedLoad)
Else
    // Step 5: Maintain or Decrease Signal Timing
    adjust_traffic_signals('decrease', PredictedLoad)
End If
// Step 6: Priority Vehicle Handling
If emergency_vehicle_detected() then
    grant_priority_to_emergency_vehicle()
// Step 7: Feedback Loop for Continuous Adjustment
While system is active do
    Collect new traffic data
    TrafficData = collect_traffic_data()
    Analyze new traffic data
    PredictedLoad = analyze_traffic_patterns(TrafficData)
    Adjust signal timings based on new load prediction
    adjust_traffic_signals(PredictedLoad)
End While
// Step 8: System Monitoring and Evaluation
Monitor system performance
Evaluate traffic flow improvements and make manual adjustments if needed
END
```

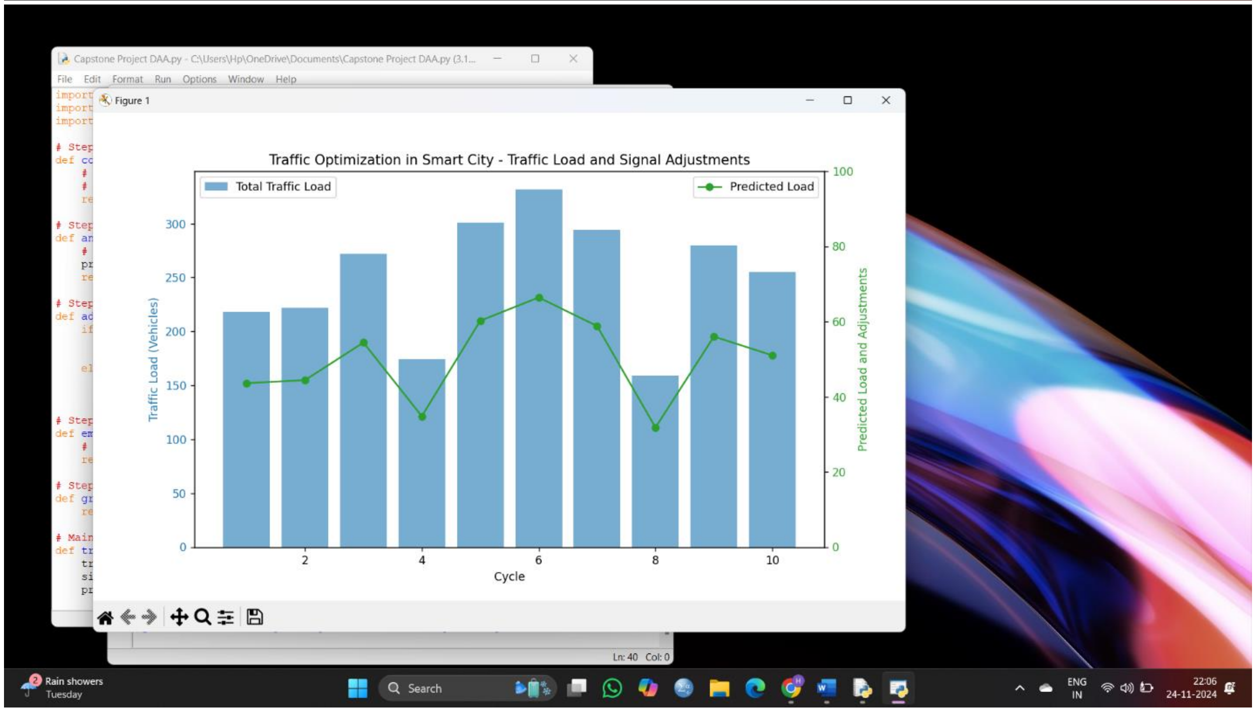
Actual Code:

```
import random
import time
import matplotlib.pyplot as plt
# Step 1: Simulate real-time data collection
def collect_traffic_data():
    # Simulate traffic data: vehicle count at different intersections
    # For simplicity, we use random values between 0 and 100 for traffic density
    return [random.randint(0, 100) for _ in range(5)]  # 5 intersections
# Step 2: Analyze traffic patterns and predict congestion
def analyze_traffic_patterns(traffic_data):
    # Predict traffic load (simple average of all intersection vehicle counts)
    predicted_load = sum(traffic_data) / len(traffic_data)
    return predicted_load
# Step 3: Adjust traffic signals based on predicted load
def adjust_traffic_signals(predicted_load, average_load=50):
    if predicted_load > average_load:
        # Increase green light duration for heavy traffic
        return "Increase green light duration"
    else:
        # Decrease or maintain normal signal timings
        return "Maintain or decrease green light duration"
# Step 4: Check for emergency vehicles (simulated)
def emergency_vehicle_detected():
    # Random chance for an emergency vehicle to be detected
    return random.choice([True, False])
# Step 5: Grant priority to emergency vehicles
def grant_priority_to_emergency_vehicle():
    return "Grant priority to emergency vehicle"
# Main program with graphing capabilities
def traffic_optimization_system():
    traffic_data_history = []
    signal_adjustment_history = []
    predicted_load_history = []
    # Simulating the system running for 10 cycles
    for cycle in range(10):
        # Step 1: Collect real-time traffic data
        traffic_data = collect_traffic_data()
        print(f"Cycle {cycle+1} - Collected traffic data: {traffic_data}")
        # Step 2: Analyze traffic data to predict congestion
        predicted_load = analyze_traffic_patterns(traffic_data)
        print(f"Cycle {cycle+1} - Predicted load: {predicted_load}")
        # Step 3: Adjust traffic signals based on the predicted load
        signal_adjustment = adjust_traffic_signals(predicted_load)
        print(f"Cycle {cycle+1} - Traffic signal adjustment: {signal_adjustment}")
        # Step 4: Handle emergency vehicle priority
        if emergency_vehicle_detected():
            priority_action = grant_priority_to_emergency_vehicle()
            print(f"Cycle {cycle+1} - Priority action: {priority_action}")
        # Store data for graphing
        traffic_data_history.append(traffic_data)
        signal_adjustment_history.append(signal_adjustment)
        predicted_load_history.append(predicted_load)
        # Simulate waiting for next cycle (5 seconds)
        time.sleep(1)
    # After all cycles, plot the graph
    plot_graph(traffic_data_history, predicted_load_history, signal_adjustment_history)
# Function to plot the traffic data and signal adjustments
def plot_graph(traffic_data_history, predicted_load_history, signal_adjustment_history):
    # Create subplots for traffic data and signal adjustment history
    fig, ax1 = plt.subplots(figsize=(10, 6))
    # Plot traffic data (bar graph)
    ax1.set_xlabel('Cycle')
    ax1.set_ylabel('Traffic Load (Vehicles)', color='tab:blue')
    ax1.bar(range(1, len(traffic_data_history) + 1), [sum(data) for data in traffic_data_history],
color='tab:blue', alpha=0.6, label="Total Traffic Load")
    ax1.tick_params(axis='y', labelcolor='tab:blue')
    # Create a second y-axis for the predicted load and signal adjustments
    ax2 = ax1.twinx()
    ax2.set_ylabel('Predicted Load and Adjustments', color='tab:green')
    ax2.plot(range(1, len(predicted_load_history) + 1), predicted_load_history, color='tab:green',
label="Predicted Load", marker='o')
    ax2.set_ylim(0, 100)
    ax2.tick_params(axis='y', labelcolor='tab:green')
    # Add a legend and show the plot
    ax1.legend(loc='upper left')
    ax2.legend(loc='upper right')
    plt.title("Traffic Optimization in Smart City - Traffic Load and Signal Adjustments")
    plt.show()
# Start the traffic optimization system
traffic_optimization_system()
```

Output Screen Shots:



```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: C:\Users\Hp\OneDrive\Documents\Capstone Project DAA.py =======
Cycle 1 - Collected traffic data: [3, 60, 15, 44, 96]
Cycle 1 - Predicted load: 43.6
Cycle 1 - Traffic signal adjustment: Maintain or decrease green light duration
Cycle 2 - Collected traffic data: [7, 35, 85, 0, 95]
Cycle 2 - Predicted load: 44.4
Cycle 2 - Traffic signal adjustment: Maintain or decrease green light duration
Cycle 2 - Priority action: Grant priority to emergency vehicle
Cycle 3 - Collected traffic data: [15, 80, 68, 45, 64]
Cycle 3 - Predicted load: 54.4
Cycle 3 - Traffic signal adjustment: Increase green light duration
Cycle 3 - Priority action: Grant priority to emergency vehicle
Cycle 4 - Collected traffic data: [26, 16, 28, 94, 10]
Cycle 4 - Predicted load: 34.8
Cycle 4 - Traffic signal adjustment: Maintain or decrease green light duration
Cycle 4 - Priority action: Grant priority to emergency vehicle
Cycle 5 - Collected traffic data: [61, 45, 73, 37, 85]
Cycle 5 - Predicted load: 60.2
Cycle 5 - Traffic signal adjustment: Increase green light duration
Cycle 5 - Priority action: Grant priority to emergency vehicle
Cycle 6 - Collected traffic data: [96, 15, 82, 78, 61]
Cycle 6 - Predicted load: 66.4
Cycle 6 - Traffic signal adjustment: Increase green light duration
Cycle 7 - Collected traffic data: [47, 79, 74, 49, 45]
Cycle 7 - Predicted load: 58.8
Cycle 7 - Traffic signal adjustment: Increase green light duration
Cycle 8 - Collected traffic data: [45, 7, 56, 44, 7]
Cycle 8 - Predicted load: 31.8
Cycle 8 - Traffic signal adjustment: Maintain or decrease green light duration
Cycle 9 - Collected traffic data: [68, 24, 66, 49, 73]
Cycle 9 - Predicted load: 56.0
Cycle 9 - Traffic signal adjustment: Increase green light duration
Cycle 9 - Priority action: Grant priority to emergency vehicle
Cycle 10 - Collected traffic data: [4, 62, 83, 18, 88]
Cycle 10 - Predicted load: 51.0
Cycle 10 - Traffic signal adjustment: Increase green light duration
```



Complexity Analysis:

Time Complexity Analysis:

Let's analyze the time complexity of each part of the program:

1. Data Collection (collect_traffic_data):
    - Time Complexity: O(n)
    - The function generates a list of random values for n intersections. If there are n intersections, the time complexity is proportional to the size of the list, which is O(n).
2. Traffic Analysis (analyze_traffic_patterns):
    - Time Complexity: O(n)
    - The function calculates the average of the traffic data, which requires summing up all the values. The time complexity of summing n values is O(n).
3. Signal Adjustment (adjust_traffic_signals):
    - Time Complexity: O(1)
    - This function simply compares the predicted load with the average load and adjusts the signal accordingly. This is a constant-time operation, so the time complexity is O(1).
4. Emergency Vehicle Check (emergency_vehicle_detected):
    - Time Complexity: O(1)
    - The function randomly chooses whether an emergency vehicle is detected. This is a constant-time operation (O(1)).
5. Priority Handling (grant_priority_to_emergency_vehicle):
    - Time Complexity: O(1)
    - The function performs a simple task, which is giving priority to an emergency vehicle. Hence, it is O(1).
6. Graph Plotting (plot_graph):
    - Time Complexity: O(m)
    - The plot_graph function iterates over the history of traffic data, predicted loads, and signal adjustments to plot the graphs. If m is the number of cycles (10 in this case), this function will perform a constant amount of work for each cycle. Hence, the time complexity is O(m).

Overall Time Complexity:

The main program is running in a loop that performs the following tasks:

- Collecting traffic data: O(n)

- Analyzing traffic data: O(n)
- Adjusting signals: O(1)
- Handling emergency vehicles: O(1)
- Storing the data for graphing: O(1)

The overall time complexity for each cycle is dominated by the traffic data collection and analysis steps, which are both O(n). The graph plotting at the end of the simulation is O(m), where m is the number of cycles.

- Time Complexity per Cycle: O(n) (since data collection and analysis dominate)
- Total Time Complexity (for k cycles): O(k * n + m) where k is the number of cycles (10 in this case) and n is the number of intersections.

Given that m is constant (10 cycles) and n is the number of intersections, the overall complexity simplifies to:

Total Time Complexity: O(k * n)

Space Complexity Analysis:

1. Data Storage:
   - We store the traffic data, predicted load, and signal adjustment history for k cycles.
   - The space required for each list (traffic_data_history, predicted_load_history, signal_adjustment_history) is O(k).
2. Graph Plotting:
   - The space used for plotting the graph depends on the number of cycles, k, and the number of data points in each cycle (n). However, this does not significantly add to the space complexity beyond the O(k) storage required for the history.

Overall Space Complexity:

- The space complexity is driven by the storage of data over k cycles. Since we store n values for each of the three lists (traffic data, predicted load, and signal adjustments), the space complexity is:

Space Complexity: O(k * n).

Summary of Complexity:

- Time Complexity: O(k * n) where k is the number of cycles and n is the number of intersections.
- Space Complexity: O(k * n) for storing the traffic data, predicted loads, and signal adjustments.

<u>Conclusion:</u>

The initiative, Enhancing Traffic Management in a Smart City, showcases a methodical strategy to tackle urban traffic congestion by utilizing real-time data, predictive modeling, and adaptive traffic control systems. By incorporating these elements into a continuous feedback mechanism, the system effectively distributes traffic loads at intersections, minimizes delays, and gives priority to emergency vehicles, thus improving overall urban transportation.

The execution of such a system could significantly enhance urban living by shortening travel times, reducing fuel usage, and decreasing greenhouse gas emissions. The incorporation of data visualization through charts further aids the decision-making process by offering actionable insights into traffic trends and system effectiveness.

This project sets the stage for future developments, such as:

Integrating artificial intelligence and machine learning for enhanced traffic forecasting.

Broadening the system to encompass pedestrian flow management and optimization of public transport.

Facilitating IoT connectivity for smooth interactions between vehicles, traffic signals, and management systems.

In summary, the suggested solution not only deals with current traffic issues but also aligns with long-term objectives aimed at creating sustainable and intelligent urban infrastructures, paving the way for genuinely smart cities.