Project Report on Data Structures and Algorithms

Smart Queue-Based Traffic System Simulation

Assignment Number 1

Submitted by:-Binayak Dhungana
Roll no:- : 23
Group:- CS-I (II/I)

Submitted to:- Mr Rupak Ghimire Sir
DOCSE, KU, Kavre

Table of Contents

Table of Contents ii

1. Introduction 1

1.1 Background 1

1.2 Problem Statement 1

1.3 Objective 1

2. System Requirement and Summary 2

2.1 System Requirement 2

2.2 Summary of Work 3

3. Data Structure 4

4. Algorithm used for traffic control 5

4.1 Hysteresis Control Logic 5

4.2 Traffic Light Switching Algorithm 6

4.3 Dynamic Duration Calculation 6

5. Time Complexity Analysis 7

5.1 Traffic Control Decision 7

5.2 Vehicle Generation 7

5.3 Simulation Animation Traversal 7

6. Conclusion and Future Scope 8

6.1 Conclusion 8

6.2 Future Scope 8

INTRODUCTION

1.1 Background

Urban traffic congestion remains a critical problem. Traditional fixed-time traffic controllers (often using a simple Round-Robin schedule) fail to adapt to real-time traffic volume. This inefficiency causes unnecessary delays when a lane is given a green light with no waiting cars, or when a heavily congested lane receives insufficient service time.

1.2 Problem Statement

The objective of this project was to simulate an advanced traffic control system that dynamically adjusts signal timing based on real-time vehicle density, and, critically, implements a Hysteresis-based Priority Logic to effectively manage critical high-density lanes.

1.3 Objectives

- To implement a visual, multi-threaded simulation of a 4-way intersection using Pygame.
- To utilize custom Queue and Priority Queue data structures to manage vehicle flow and traffic light phases.
- To implement a Dynamic Green Time algorithm based on the average queue length of participating lanes.
- To implement a robust Hysteresis Control algorithm for the designated priority lane (AL2) to prevent oscillation and effectively clear major congestion.

2. SYSTEM REQUIREMENTS AND SUMMARY OF WORK

2.1 System Requirements

Component	Specification
Programming Language	Python 3.x
GUI Library	Pygame (Used for visualization, rendering, and event handling)
Key Modules	threading (For concurrent processing), collections.deque, heapq (For Priority Queue)
IDE/Text Editor	VS Code / PyCharm / IDLE

2.2 Summary of Work

A real-time, multi-threaded traffic simulation was developed using Python and Pygame. The system models a 12-lane junction (3 lanes per direction) and employs an Adaptive Traffic Controller that prioritizes vehicle throughput over fixed cycles.

Key Features of Implementation:

- **Multi-Threading:** Utilizes separate threads (generator, light_changer, traversal) for concurrent generation, logic control, and vehicle movement.
- **Dynamic Lane Mapping: Vehicles are spawned into specific lanes (L2 for straight/right, L3 for left turns).**
- **Hysteresis Priority Control:** Lane AL2 is designated as the priority lane. Green time allocation is dynamically calculated and overridden when AL2 exceeds the Trigger Threshold (10 vehicles) and remains active until the count drops below the Deactivation Threshold (5 vehicles).
- **Dynamic Phase Timing:** Green light duration for all phases is calculated based on current traffic volume.

3. DATA STRUCTURES

The simulation heavily relies on custom and built-in Python data structures to achieve efficient vehicle management and traffic phase prioritization.

Data Structure	Implementation in Code	Purpose and Usage
Custom Queue	class Queue (using internal Python list)	**Lane Vehicle Management:** Stores vehicle IDs for each of the 12 lanes (lane['AL2'].enqueue(vehicle_id)). Provides $O(1)$ size lookups for the traffic controller.
Priority Queue	class LanePriorityQueue (using heapq module)	**Traffic Phase Selection:** Used within the light_changer to push traffic phases (e.g., "AC_PRIORITY", "BD") along with a numerical priority, ensuring the highest priority phase is always selected first.
Dictionary of Queues	lane = {"AL1": Queue(), ...}	**Lane State Storage:** The central repository for the entire junction's state. Allows $O(1)$ access to any lane's queue for size calculation and vehicle dequeueing.
Dictionary of Objects	visual_vehicles = {vehicle_id: Vehicle_Object}	**Visual Rendering & Movement:** Maps a vehicle's unique ID to its Vehicle object, allowing the main Pygame loop to efficiently update, rotate, and draw moving cars.

4. ALGORITHM USED FOR TRAFFIC CONTROL

The traffic control logic is executed by the concurrent light_changer() function and utilizes a Dynamic Green Time calculation wrapped in a **Hysteresis-based Priority Override** mechanism.

4.1 Hysteresis Control Logic

Hysteresis prevents "traffic light flickering" by requiring a significant difference between the activation and deactivation points.

Component	**Code Implementation**	**Value**
Trigger Threshold	PRIORITY_TRIGGER_THRESHOLD	10 vehicles
Deactivation Threshold	PRIORITY_DEACTIVATE_THRESHOLD	5 vehicles
Control Function	is_priority_active()	**Logic:** If AL2 size > 10, set priority ON. If AL2 size ≤ 5, set priority OFF.

4.2 Traffic Light Switching Algorithm (light_changer)

The controller operates in phases selected by a Priority Queue:

Phase Selection	Priority Value	Lane Serviced	Green Time Formula
AC_PRIORITY	0 (Highest)	AL2, AL3, CL2, CL3	**Priority Time:** $\text{AL2 size} \times 1s$
AC (Normal)	1	AL2, AL3, CL2, CL3	**Normal Time:** $\max(8s, \text{Avg. Waiting Cars} \times 1s)$
BD (Normal)	2	BL2, BL3, DL2, DL3	**Normal Time:** $\max(8s, \text{Avg. Waiting Cars} \times 1s)$

4.3 Dynamic Duration Calculation

Green time (T) is calculated to match the expected vehicle throughput for the phase:

$$T = \max(\text{MinGreenTime}, (\text{VehiclesToServe} \times \text{TIME\_PER\_VEHICLE}))$$

Where VehiclesToServe is calculated using the helper function $\text{calculate\_vehicles\_to\_serve}()$ which finds the average queue length of the relevant lanes.

5. TIME COMPLEXITY ANALYSIS

5.1 Traffic Control Decision (light_changer and is_priority_active)

- **Complexity:** $O(1)$
- **Explanation:** The traffic controller's core decision involves accessing the size of specific queues (lane["AL2"].size()) and performing a fixed number of arithmetic operations (comparison, maximum, average). The size operation for a Python list/custom queue is $O(1)$.

5.2 Vehicle Generation (generator)

- **Complexity:** $O(C)$
- **Explanation:** The generator iterates through a fixed number of lanes ($C=8$ lanes, L2 and L3). For each lane, it performs a fixed number of $O(1)$ operations (enqueue). Since C is a constant, the complexity is $O(1)$ or $O(C)$.

5.3 Simulation Animation and Traversal (update in Vehicle and traversal thread)

- **Traversal Complexity:** $O(1)$
    - The traversal thread checks the traffic light state and performs a fixed number of $O(1)$ dequeue operations if the light is green. The number of checks is constant.
- **Animation Complexity:** $O(N)$
    - The main Pygame loop iterates through the visual_vehicles dictionary (where N is the number of active vehicles). For each vehicle, it performs an $O(1)$ update to move it along its pre-calculated path. Unlike simple 2D collision detection, no inner loop is required, keeping the animation smooth and efficient.

6. CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

The project successfully simulated an adaptive traffic junction by strategically applying Data Structures and Algorithms. The use of custom **Queues** for vehicle management and the implementation of a **Hysteresis-based Priority Logic** demonstrated an effective solution to the common urban problem of traffic congestion. This approach ensures that green light time is proportional to actual demand, significantly improving flow efficiency, especially under high-stress conditions.

6.2 Future Scope

This simulation provides a strong foundation for future development:

- **Advanced Collision Modeling:** Implement full lane-by-lane proximity checks within the traversal function to ensure safety gaps are maintained, rather than relying solely on the time buffer (last_move_time).
- **Weighted Priority:** Assign different weights to vehicles (e.g., buses, trucks) and use these weights in the queue length calculation to adjust priority beyond a simple count.
- **Multi-Agent Learning:** Incorporate a simple AI agent that learns the optimal trigger and deactivation thresholds based on historical congestion data, further optimizing the Hysteresis parameters.