CSE6730 PROJECT-1 CHECKPOINT

CHANDIM CHATTERJEE, SEYED PARSA BANIHASHEMI & PIYUSH SAO

Contents

1. Project Description

This project is a discrete event traffic simulation of a section of Peachtree street (from 10th street to 14th street) from 4pm to 6pm. The simulation will analyse the rush-hour behavior of traffic in the above-mentioned road sections. The simulation is based on a stochastic model that models the incoming and outgoing cars using a random number generator based on data collected by a research by the National Institute for Advanced Transportation Technology titled "Improved Simulation of Stop Bar Driver Behavior at Signalized Intersections". The main goals of the project is to capture the distribution of vehicle travel times as they traverse the stretch of Peachtree Str and possibly store this data in a text file for further data analysis.

The stretch of Peachtree Street that is modeled is shown in Figure 1. This stretch has five intersections, each with 4 adjoining roads coming from the North, South, East

and West. The street throughout the stretch is aligned from North to South and has two lanes moving in both directions. Four of the five intersections are run using traffic lights whose timings are predictable and known while one of them is governed by stop signs. Cars can enter this stretch from any of the intersections or can be coming North on Peachtree at the 10th Street intersection or be moving south on Peachtree at the 14th Street intersection. At each intersection, the cars can move towards their final destination only when they have a green signal or the right of way. The incoming cars spend time averse through distances along the stretch of the road and queue up before intersections.

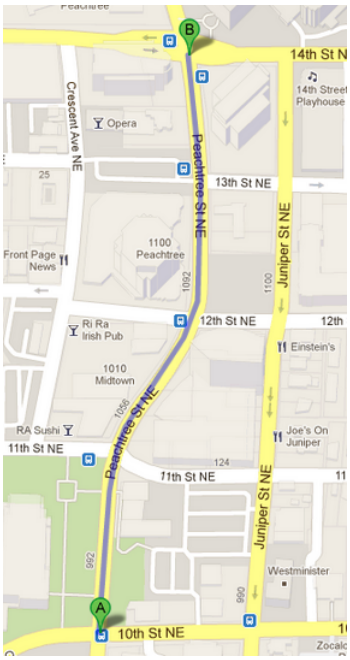


FIGURE 1.1. Satelite view of W. PeachTree Street

The data received from the National Institute research is used to find the distribution of car arrival times and measure average travel times for car through different stretches of the road. This data is also used to stochastically assign final destinations of arriving cars.

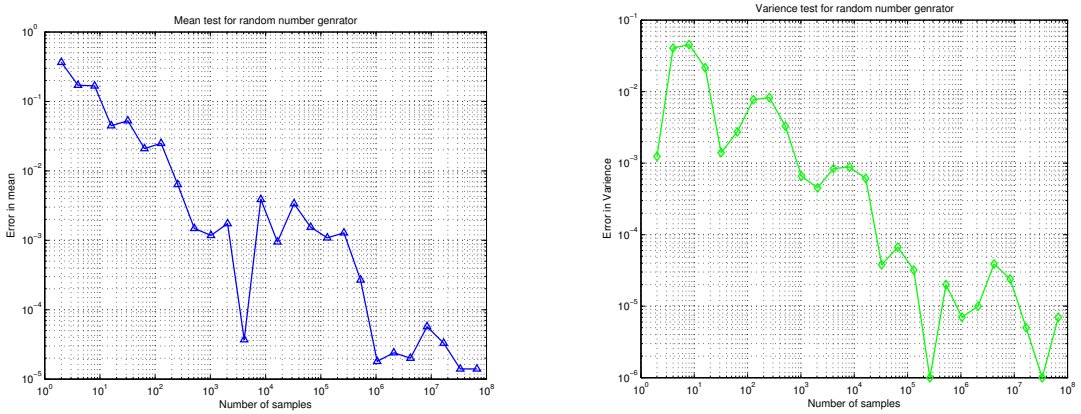
Some of the assumptions made in the System Under Investigation while creating the model are listed below:

(1) Once cars move out of the stretch (North of 14th street or South of 10th street), they can move freely to their final destination and there is no congestion in the exit points that may stop vehicles from proceeding. This is done to simplify the model as the main goal is to predict peak hour traffic behavior and traversal times only within the stretch

(2) All vehicles try to take the shortest route and move to the correct lanes to proceed to the final destination. This may not always be true but a high percentage of peak hour traffic vehicles know the stretch well enough to make the correct decisions to proceed to final destination
(3) There are no emergency or regulatory vehicles in action during this time.
(4) There are no traffic accidents that may create congestion in the model.
(5) Pedestrian traffic is not considered in this model and it is assumed that they cross the road only during red signal and do not affect vehicle traffic. Their effect on the stop light intersection is also ignored and may be revisited in future iterations
(6) It is assumed that cars move at similar uniform speeds through the stretch during this time. This is assumed to be true because during peak hours, the effect of difference in acceleration of different cars is not too pronounced due to high congestion. This is another assumption that may be revisited in future iterations

2. Random Number Genrator

In heart of any simulation engine lies a random number generator. Hence, designing a pesudo random number generator, was an important consideration for the project. We would ideally like random number generator to have a large period. It should be reasonably fast so that overhead of computing random numbers is not high. Additionally, we would like it to produce random numbers in accordance to distribution. We chose Lehmer's algorithm [REF] for generating random numbers for it is fast and well tested. Various parameters to lehmer's algorithm have been studied for lehmer's algorithm, we after trying a couple of possibilties,we choose $n = 2^{31}-1$, $g = 7^5$. This combination is also known as MINSTD. It has a full cycle of $n = 2^{31} - 1 = 2147483645$, which we explicitly verify by running it for full cycle. We verify that with increasing number of trials, mean of output of the random numbers converge to 0.5 and varience converges to $\frac{1}{12}$, for uniform distribution[REF:beautiful testing].



(A) (B)

FIGURE 2.1

In addition to that, we use Chi Square testing [REF], to test uniform distribution hypothesis of our random number generator.

Number of Buckets	Number of trials	χ^2	Probability
10	10000000	7.06	0.6308
100	10000000	89.18	0.75
1000	10000000	930.80	0.9390
10000	10000000	9675.33	0.9895
100000	10000000	98917.66	0.9923
1000000	10000000	994092	0.9999

TABLE 1. Chi Square test for our Random Number Generator

3. The calendar queue

Discrete time simulation requires handling of large number of events. All the events are scheduled with a time stamp (More on this at 9). Task of priority queue is ensure store all the events that are scheduled and evict them from list based on their time stamp, thereby ensuring causality of the simulation. The number of events in the simulation grows to a large value quickly, which requires the priority queues to handle such large number of events efficiently. We choose calender queue [REFF] which the original paper claimed to be $\mathcal{O}(1)$. In theory its not really an $\mathcal{O}(1)$ algorithm, but for range of N we need it performs very well (3.1).

Calender queue achieves this by dividing the main list into k small lists , which each of them we call a bucket. Bucket i allows values of time t, given as follows

$$\left\lfloor \frac{t - P\left\lfloor \frac{t}{P} \right\rfloor}{W} \right\rfloor = i$$

Where P is one period of the calender and W is width of the window. Since each lists remains small enough and figuring out which list remains a constant time operation, which allows almost constant time popping from the queue. Original paper [REFF] described resizing the calender so that when number

The events list is managed by a calendar queue data structure, with constant bucket sizes of size 0.1s for this checkpoint. The bucket size is easy to modify and can be tested for optimal performance. Each bucket contains a linked list of events and the length of the linked list is tried to be kept lower than 5 for now. The calendar queue fits very well for our purpose, since it provides O(1) access time with high probability. This abstract data structure is very similar to a hash table with an implemented chaining conflict resolution mechanism.

3.1. **Performance of calender queue.** To test the performance of our calender queue implementation, we use hold time model. We compare the Calender queue with linked list (which is also special case of calender queue with number of buckets equal to one). The performance comparison is depicted in 3.1

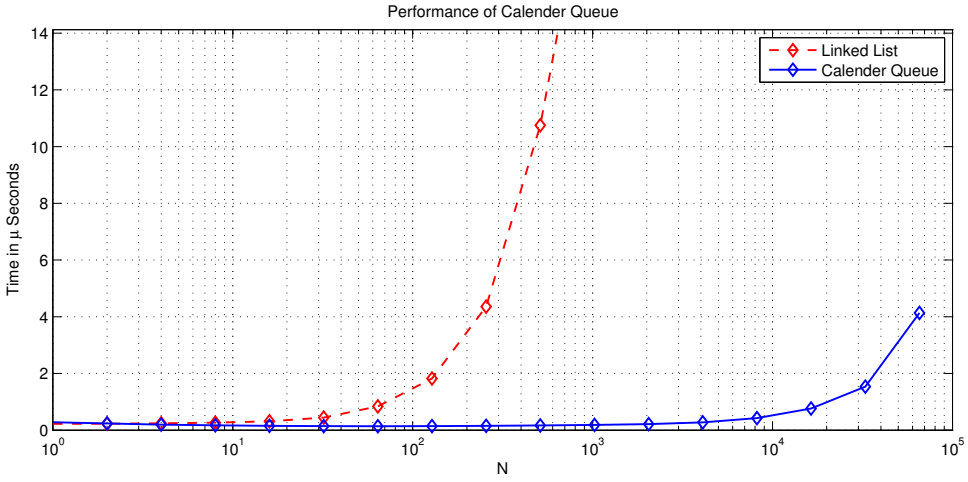


FIGURE 3.1. Performance comparison of Calender queue with linked list

4. Conceptual Model

The conceptual model used is a simple queueing model where each all the road patches between intersections and entering into and exiting from the intersections are modeled as queues which are created by linked lists (using C++ STL std::list). The simulator program also uses a priority queue to store the future event list and the queue is created using a simple hashing abstract data structure called calendar queue. This provides a data access and insertion time complexity of nearly O(1) with high probability. Priority queue models also rely heavily on random number generators for simulation of events and a custom random number generator will be written in the future.

Entities and Events:

4.1. **Simulator and the Event handler.** Events:

- StopAt
- Run
- Schedule

The simulator is the main engine of the code.The simulator schedules the events and by doing this the events are added in the priority queue (implemented using a calendar queue) sorted by their times. The simulator can schedule when the simulation should stop and when it's run is called it keeps popping events from its priority queue until the stop event is encountered or there are no more events in the queue.

The events are created in a templated manner so that the event call handler functions can easily be called as member functions of classes. This can be observed easily in more detail by checking the schedule function in the code.

4.2. **Vehicles.** The consumer objects in our model are the vehicles passing through the road network. The vehicles are a class , with attributes, such as ID, start time, end time, current direction, last queue, and, final destination. The final destinations are randomly generated when the vehicle is generated. The vehicles also keep track of their start times and end times and are added to an exit queue after their pass through the stretch. The start times and end times are used to generate the essential data/metric for this project

4.3. **VehicleQueues.** The queues are linked lists, containing components of vehicle type. Queues are treated in a first-in first-out fashion.

4.4. **Traffic Lights.** Events:

- cyclestate

The traffic lights are attributes of the intersections with signal. They have a "Current phase" variable. Whenever a "cyclestate" event is scheduled, the traffic light changes state to the next and schedules a change of a "changeSignalTrigger" event that triggers traveling of cars that are in relevant queues. There are four TrafficLights for each intersection with signal.

4.5. **6. Intersections.** Events:

- changeSignalTrigger
- AddtoQueue
- VehiclePass
- VehicleDeparture

We have two types of intersections: with signals and without signals. The two types of intersection classes inherit from an abstract Intersection base class. Intersections are the main event scheduler in the program. Different methods in the intersection classes schedule other methods for the future execution by calling the simulator Schedule function. The intersection contains the routing table which is used to direct vehicles towards their final destination. The Intersection class contains pointers to different VehicleQueues that are adjoining it. A set of busy flags within the class will also represent the state of the intersection and govern whether cars can enter it. The AddtoQueue, VehiclePass and VehicleDeparture are used to depict the motion of a car across the intersection and being moved from one queue to another.

4.6. **Topology.** The topology class contains object classes that represent the whole stretch of roads including intersections and vehiclequeues. It also keeps track of an exit queue containing all the cars that have completed their traversal of the stretch. Most of the randomness in our program will go into this class as all the initial vehicle addition events are scheduled here (not implemented completely yet, but is the next step in the development process).

4.6.1. *Simulation Model.* Classes, Data Structures, and methods.

(1) VehicleClass: The most basic data structure is a class for individual vehicles. It contains the following attributes:
   - ID: The unique tag to identify each vehicle.
   - StartTime: The time of arrival of the vehicle into the system.
   - End Time: The time that the vehicle reaches its destination.
   - Destination: The ID of the particular node that the vehicle must reach.
   - Length: The length of the vehicle. It is applied to calculate the length of the queue
   - Current Direction: the direction which the intersection directs the vehicle while the VehiclePass is executed.
   - Last Queue: The last queue the vehicle was a member of before entering the intersection.

(2) VehicleQueue: is a linked list of type "queue" of VehicleClass.

(3) TrafficLight: contains several attributes
   (a) Data structures:
      - Signal Type: Indicates whether a traffic light is a 3-phase, or a 6-phase light.
      - Signal Timings: A set of variables containing the time for each phase.
      - Parent intersection: used for issuing a "changeSignalTrigger" event.
      - CurrentState: Indicates the current phase.
   (b) Methods:
      - cyclestate: This method performs a change of phase for the traffic light. All "cyclestate" events should be scheduled in the beginning of the simulation.

(4) Intersection: is the base class for "Intersection with Signal" and "Intersection Without Signal" classes. Its attributes are:
   (a) Data Structures:
      - ID: the unique identification number for intersections.
      - Busy indicators: boolean flags to specify whether the intersection is busy.
      - routingtable: is an array that provides a direction to each vehicle judging based on the vehicle's final destination.
      - Incoming Queues: There are 4 vehicle queues incoming each intersection, and a global ExitQ that each intersection has the pointer to it. Whenever a vehicle leaves the intersection, it is added to that queue. ExitQ is going to be used to analyse the system in the future.
      - Neighbor intersections: The pointer to the neighbor intersections is included in each intersection, so each intersection can schedule an arrival (addToQueue) event for the others once the vehicle departs from it.
   (b) Methods:
      - Add to queue: This is a "virtual" method that is overridden by other classes inheriting this class.

(5) Intersection with signal: This class inherits from the Intersection class and has these additional parts:
   (a) Data Structures:
      - Traffic Lights: TrafficLight type
   (b) Methods:
      - VehiclePass: Sets the intersection to busy and schedules a departure event, using the routingTable.
      - VehicleDeparture: Sets "busy" to "false", schedules joining the next queue, or exiting the system and adding to ExitQ. If the Light is still green, schedules the next VehiclePass for the next vehicle.
      - addVehicletoQueue: adds vehicle to the neighbor intersection's queue.
      - changeSignalTrigger: Starting scheduling VehiclePass for the proper queues that have "green light".
      - QCanGo: Takes a Queue and decides whether that Queue's light is green

(6) Intersection without signal: This class inherits from the Intersection class
   (a) Methods: (All exist in IntersectionWithLights)
      - VehiclePass
      - VehicleDeparture
      - addVehicletoQueue

(7) Topology: is a class, that contains almost all variables in the code. It is initialized in the beginning of the code and has theses attributes:
   - Intersection classes. All intersections are part of Topology
   - ExitQ: as explained above, this is a Queue that every vehicle leaving the network is added to.

(8) Simulator: runs the actual simulator and contains a calendar_queue containing future events.
(9) Event: is a class containing templated classes and functions and used to simulate any event which in the case of our program are functions belonging to any of the classes.
(10) Calendar_queue: is the data structure for handling storing the list of all "Event" classes.

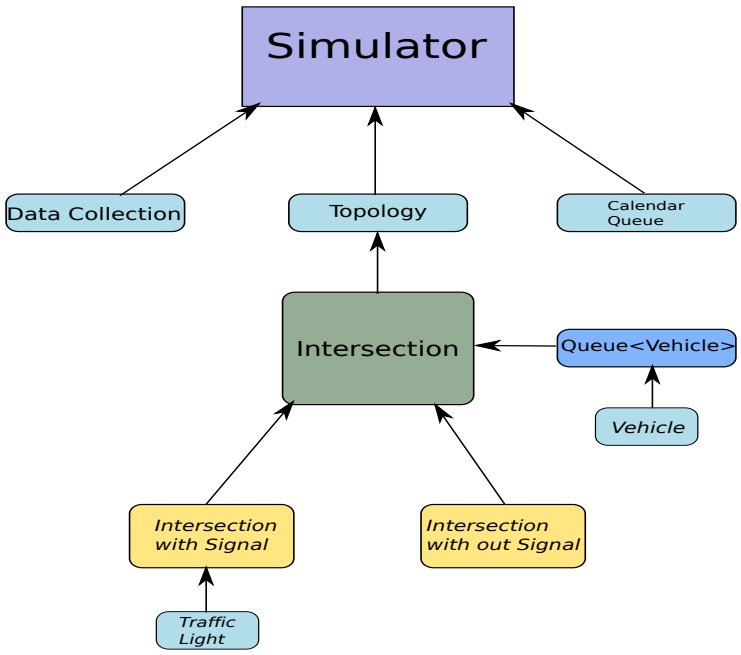


FIGURE 4.1. Class dependencies

4.7. **Typical Behavior:** The classes and methods were described in the last section. In this section, The order of execution, dependencies, and, chain of events are discussed.

- Joining the vehicles to: The events of vehicles joining the simulation system is scheduled all in the beginning of the simulation.
- cyclestate: all the change-of-state of the traffic lights are scheduled in the beginning of the simulation.
- VehiclePass can be scheduled by VehicleDeparture. So, each vehicle that departs, will schedule another vehicle's pass if the light is still green.
- changeSignalTrigger, can also start the chain of Depart-Pass. It will schedule the first VehiclePass. and the chain goes on until the light goes yellow.
- The "Stop" event, will end the simulation.
- Below, the event dependencies are demonstrated

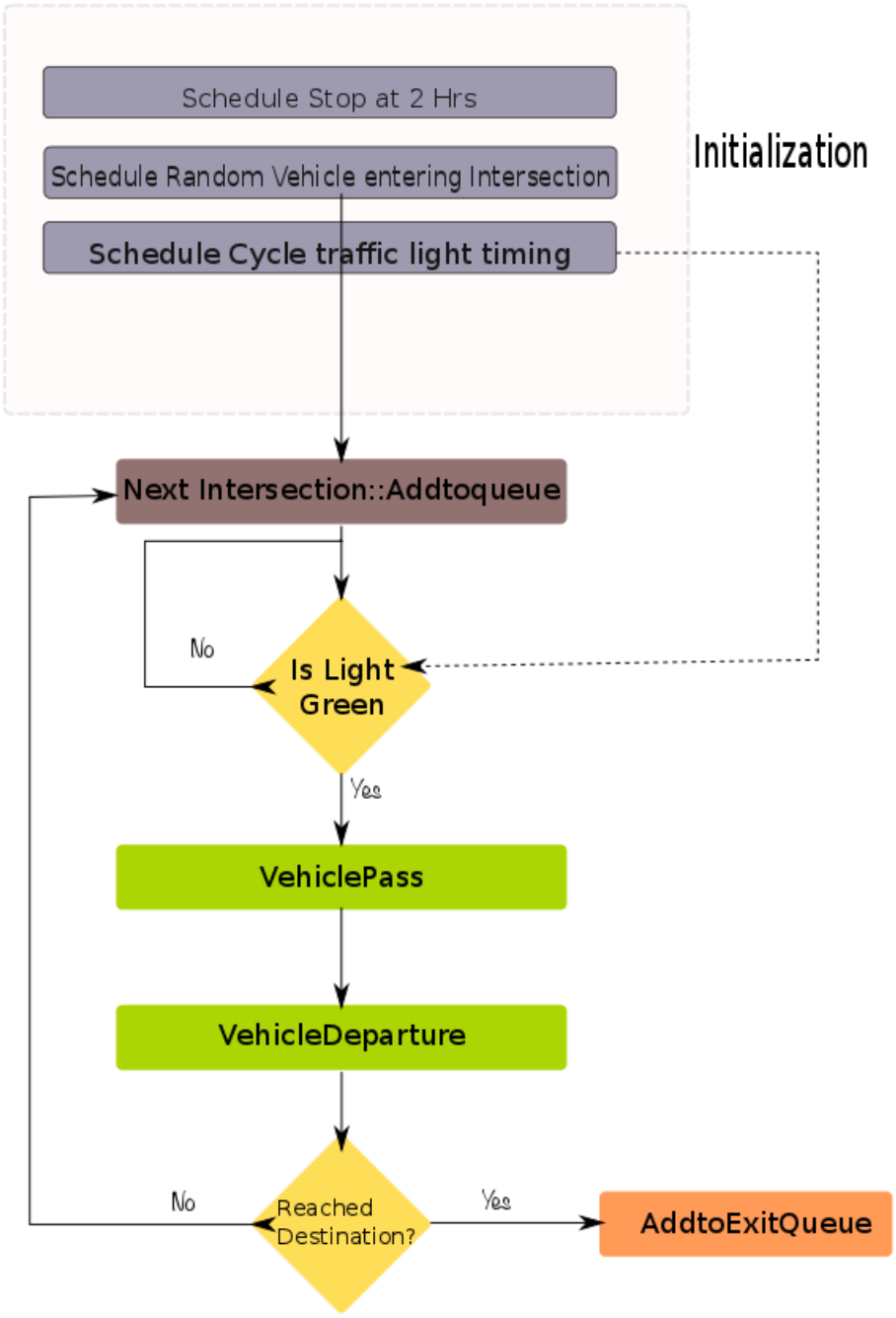


FIGURE 4.2. Event Transition Diagram

5. DIVISON OF WORK

So far, we have been able to successfully generate events that will be able self sustaining their life cycle. In order to completely simulate the W.PeachTree street, we would like to have better models of working of intersection and vehicle queues. Another

aspect of this project is to generate random data (vehicle) which statistically reflects the incoming and outgoing traffic from the W. Peachtree street. Validation of our model also remains yet another important task to be done. We would additionally like to have yet another implementation (probably splay tree/ ladder) of priority queues and see the performance of the two for this particular exercise.

Name of Task	Person responsible	Due
Priority Queues	Piyush	27 Jan
Simulator	Chandim	27 Jan
Vehicle Class	Parsa	27 Jan
Traffic lights Class	Parsa	27 Jan
Intersection Class	Parsa	27 Jan
Random Number Generator	Piyush	10 Feb
Check point report	Group activity	1 Feb
Topology Chandim	Chandim	10 Feb
Advanced Modeling of Intersection	Group activity	10 Feb
Data parsing and Initialization	Chandim	10 Feb
Advanced Modeling of Vehicle queues	Parsa	10 Feb

6. References

- S. K. Park and K. W. Miller (1988). "Random Number Generators: Good Ones Are Hard To Find". Communications of the ACM 31 (10): 1192–120