Transportation Simulation Final Report

CSE 6730 Group B:
Mary Benage
Vladimir Coxall
Julian Diaz-Ospina
Rajendran Mohan

February 24, 2012
Georgia Institute of Technology

Table Of Contents

1. Introduction

The problem of planning two alternate plans to route traffic from parking lots near the Georgia Dome to the major interstates (I-75/85, I-20) after the conclusion of a football game, has been presented to the CSE 6730 class as an academic example to implement an event driven Simulation Software Package.

The first task that needed to be accomplished was to lay down a conceptual model based on the problem statement which requires the use of a discrete event simulation based on a priority queue, and the use of a random number generator; and also based on the assumptions enumerated in the Conceptual Model section later on this paper.

Once the conceptual model was defined, the architecture of the package was structured based on two premises; modularity and re-usability. In order to distribute the programming load and accomplish the premises stated above, the software package was divided in four modules. The driver program called the Modeler, the Simulator Library, the Traffic Model Objects Library and the Traffic Model Events Library.

After finishing the programming and having tested the software with a simplified model[1] the two planned routes were modeled and run with the Traffic Simulator. We ran several sets of runs under both routing plans, with various iterations to check the consistency of our model and also used statistical analysis to provide a "correctness" of our average delay time.

2. Conceptual Model

[1] See the Traffic Simulator Reference Manual

The conceptual model was defined following the requirement of the use of a priority queue and a random generator, as well as the constrains and assumptions the group decided to narrow down and simplify the model. These assumptions are as follow:

- This simulation will account for the in-campus parking lots. Other parking lots are assumed to have a different evacuation route.
- There are four possible destinations for a vehicle; I75/85 North, I75/85 South, I20 East, and I20 West. The destination of the vehicles is equally distributed, as well as the location of the vehicles. In other words, at every parking there are 25% of the vehicles traveling to each of the four destinations.
- At the parking lot the vehicles are in line ready to exit the parking lot. The simulation inside the parking lot will be limited to an estimation of the time interval between vehicles exiting the parking lot.
- The evacuation routes are totally controlled by the evacuation plan. However, in order to alleviate local traffic, there will be several intersections which will allow traffic crossing the route, but no traffic other than the one coming from the parking lots will be allowed into the evacuation route.
- The average length of road required for one vehicle is assumed to be 15 feet, and the maximum speed limit will be 45 miles per hour.
- This simulation will not account for the driver's behavior. It will assume the vehicles will travel in line and there will be no passing. Also, the drivers will follow the route according to the plan.
- The vehicles will not travel over the maximum speed limit. Therefore, the time a vehicle takes to travel from the start of a road segment to the end is expected to be equal or less than the road segment length over the average speed limit.

2.1 Priority Queue

The problem of traffic routing and traffic simulation has several requirements which most naturally suggest a queuing model. Vehicles queue up to be serially served by a shared

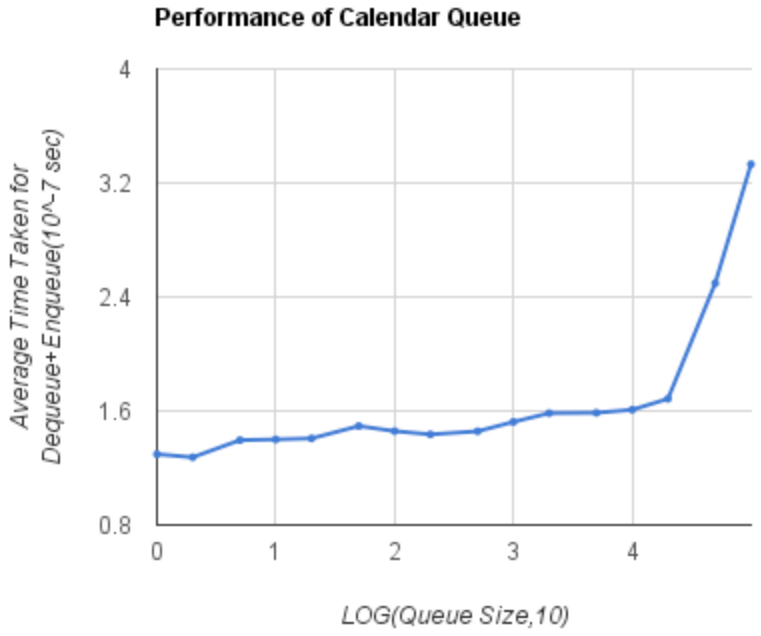
resource, and are processed with clearing times and delay times. As vehicles are cleared through queues, they advance towards an exit and then are deleted from the system.

The choice of the queueing system was based on some core functionalities that we required for our code. They were as follows:-

- Ability to store a large number of events
- Ability to return the latest event as quick as possible
- Ability to remove any particular event(not necessarily the latest) from the queue

The last factor was the most unique one that determined the actual choice of the system. Since all queueing systems have almost the same functionality with regard to the first two factors, to target any arbitrary event required would require a system that tracked events explicitly according to time. Thus, a calendar queue was chosen[8].

Once the queueing system was chosen, performance testing was done on the queue using the hold model outlined in the paper by Jones[9] using exponentially distributed timings. Repeated tests were run on the calendar queue and the following timings were obtained from the system.



Graph 1: Performance of Calendar Queue

As can be seen, nearly constant time was maintained up till a queue size of about 20,000 after which the timings have spiked. This could be attributed to cache effects since the required memory needs to be fetched from less efficient locations from the computer memory. But analysis showed that the number of events queued at any time would not reach 20000 during execution. Thus, calendar queue was deemed fit for usage in the simulation.

2.2 Random Number Generator

Our model makes use of random variates, to set properties of vehicles, service and interarrival times. Given this stochastic nature, simulation runs are representative of an analytic state – that is, stochastic simulation runs estimate the behaviour of the modelled system. A link can be established however between an experimental estimate and the analytic properties of specific classes of queuing models. Generally distributed queues (G/M/K) have been well studied and analytic relations for their properties found [5][6]. It is necessary, particularly for randomly generated vehicle properties, that the random number generator for our simulation be robust. We make use of Lehman's pseudorandom number generator as it provides uniformly distributed, independent pseudorandom numbers (IID) but is also fast. Lehmer's algorithm is well tested algorithm and has a full period. A full period ensures that it generates all the numbers in the distribution between 0 and 1. Lehmer's Algorithm is a fast calculation and easy to implement. We also do not require the full period of numbers in the Lehmer Algorithm and therefore this simple, fast, and well tested algorithm is sufficient for our model.

2.3 Physical Model

The physical problem can be stated as a number of vehicles which need to go from certain parking lots, using a shared network of road segments and intersections, to certain destinations. We have then users (vehicles) using shared resources (parking lots, road segments, intersections, and destinations) in order to reach a goal (evacuate the area). Out of this statement, five objects are identified; Vehicles, Parking Lots, Road Segments, Intersections, and Destinations.

Another object of importance to the model, although not in the physical space but in the time domain, is the Phase. A Phase can be defined as the interval of time an intersection serves vehicles coming from a particular road or group of roads. The service time for each phase has been estimated to 2.0 minutes. The intersection average crossing time defined for the turning and straight intersection crossings were 2 seconds and 1.5 seconds, respectively. The space outside a Parking Lot can be assumed as an Intersection. See Figure 1 for a physical representation of these objects.

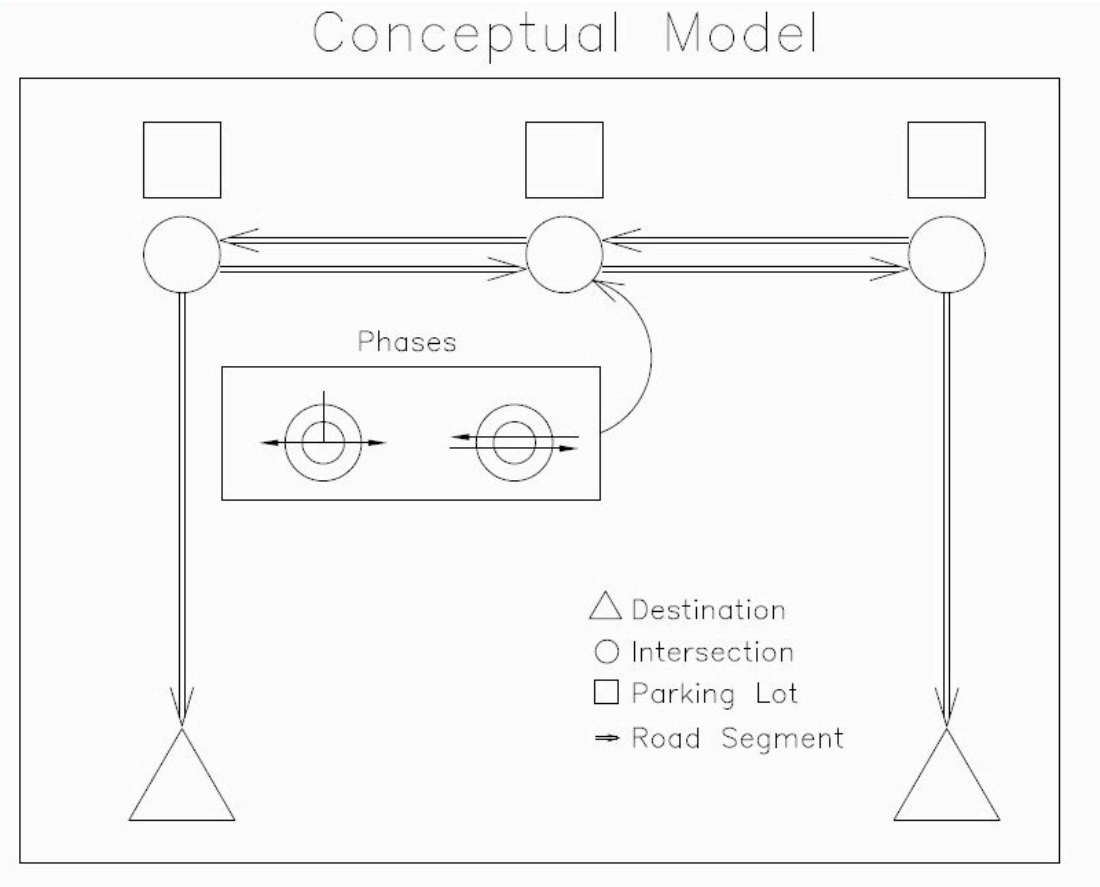


Figure 1 : Conceptual Model

The required events were deducted from the behaviour of the objects in the model. Some of them are a combination of events occurring at the same instant. i.e. When a vehicle exits a road segment (event 1) it also enters an intersection (event 2). The events used are;

Intersection Phase Change, Vehicle Arrives at Intersection, Vehicle Enters Intersection, Vehicle Exits Intersection.

2.3.1 Required Objects

The following list shows the object interfaces defined for the model. The properties enumerated are the ones required for the model. The implementation of these objects have more properties and methods required for programing.

1. **Vehicle:** The object referring to the current state of the particular vehicle
    a. **Destination**: The end location. (1-4) generated randomly on issue
    b. **Next Intersection Time**: Time at which the car reaches the next intersection (distance/speed)

2. **Parking Lot**: The object referring to the current state of a parking lot
    a. **Exits:** Total number of Exit Lanes.
    b. **Capacity**: Total number of vehicles occupying the Parking Lot.
    c. **Vehicles In**: Number of vehicles left in the parking lot
    d. **Destinations List**: List of possible destinations for the Vehicles inside.
    e. **Intersection**: Intersection outside the Parking Lot

3. **Road Segment**: The object referring to the unidirectional road segment between two Intersections. In a two way road, each way will represent a Road Segment.
    a. **Lanes**: The number of lanes in the road segment
    b. **Capacity**: Computed as the Length*Lanes/15
    c. **Minimum Travel Time**: Computed as Length/Speed
    d. **Vehicles In**: Number of Vehicles occupying the road segment.
    e. **Vehicles Ready**: A queue array with all the vehicles in the road segment in the order of their arrival

f. **Queue Parameters**: Parameters tracking head and tail of queue array
g. **Destinations List**: The list of destinations served by this Road Segment
h. **Start Intersection**: Intersection at the start of the Road Segment
i. **End Intersection**: Intersection at the end of the Road Segment

4. **Intersection**: The object referring to the intersection connecting various roads
    a. **Phases**: The number of phases(traffic intervals) at the intersection
    b. **Current Phase**: The current phase at which the intersection is at
    c. **Idle State**: True when no Vehicles waiting to cross.
    d. **Vehicles In**: Vehicles using the intersection at a given time.
    e. **Road Segments Out List**: Road Segments taking vehicles out of the Intersection.

5. **Phase:** The node from which Vehicles exit
    a. **Road Segments In List**: Road Segments using this Phase to cross the Intersection
    b. **Interval Time:** Time of duration of the Phase

6. **Destination**: The node from which Vehicles exit the area
    a. **Road Segment In**: The road reaching the exit node

2.3.2 Required Events

The required events were defined analyzing the behavior of the objects.

Events at Parking Lot:

1. **Vehicle Exit**: Occurs when a Vehicle Exits the Parking Lot. At the same time the Vehicle enters an Intersection.

Events at Intersection:

2. **Phase Change**: Occurs when duration of the current phase has elapsed, or when while in Idle state, a vehicle arrives to the intersection and request a phase change.
3. **Vehicle Enters Intersection**: Occurs when a Vehicle starts crossing the Intersection. At the same time the Vehicle has left a Parking Lot, or a Road Segment.
4. **Vehicle Exits Intersection**: Occurs when a Vehicle finishes crossing the Intersection. At the same time the Vehicle enters a Road Segment.

   Events at Road Segment:

5. **Vehicle Enters Road Segment**: Occurs when a Vehicle starts crossing the Road Segment. At the same time the Vehicle has left an Intersection.
6. **Vehicle is Ready to Exit**: Occurs when the Vehicle reaches the front row or joints the pack of vehicles in line waiting to reach the front row.
7. **Vehicle Exits Road Segment**: Occurs when a Vehicle finishes crossing the Road Segment. At the same time the Vehicle enters an Intersection or a Destination.

   Events at Destination:

8. **Vehicle Enters Destination**: Occurs when a Vehicle reaches its Destination. At the same time the Vehicle has left a Road Segment.

3. Software Architecture: Modules And Libraries

The software package was divided in four modules In order to distribute the programming load and provide modularity and re-usability to the source code. There was a need for a Driver program which we call the Modeler, a Simulator Library, an Objects Library and an Events Library. The subsections below summarize the implementation of these modules. For a more detailed explanation see the Traffic Simulator Reference Manual attached to this report.

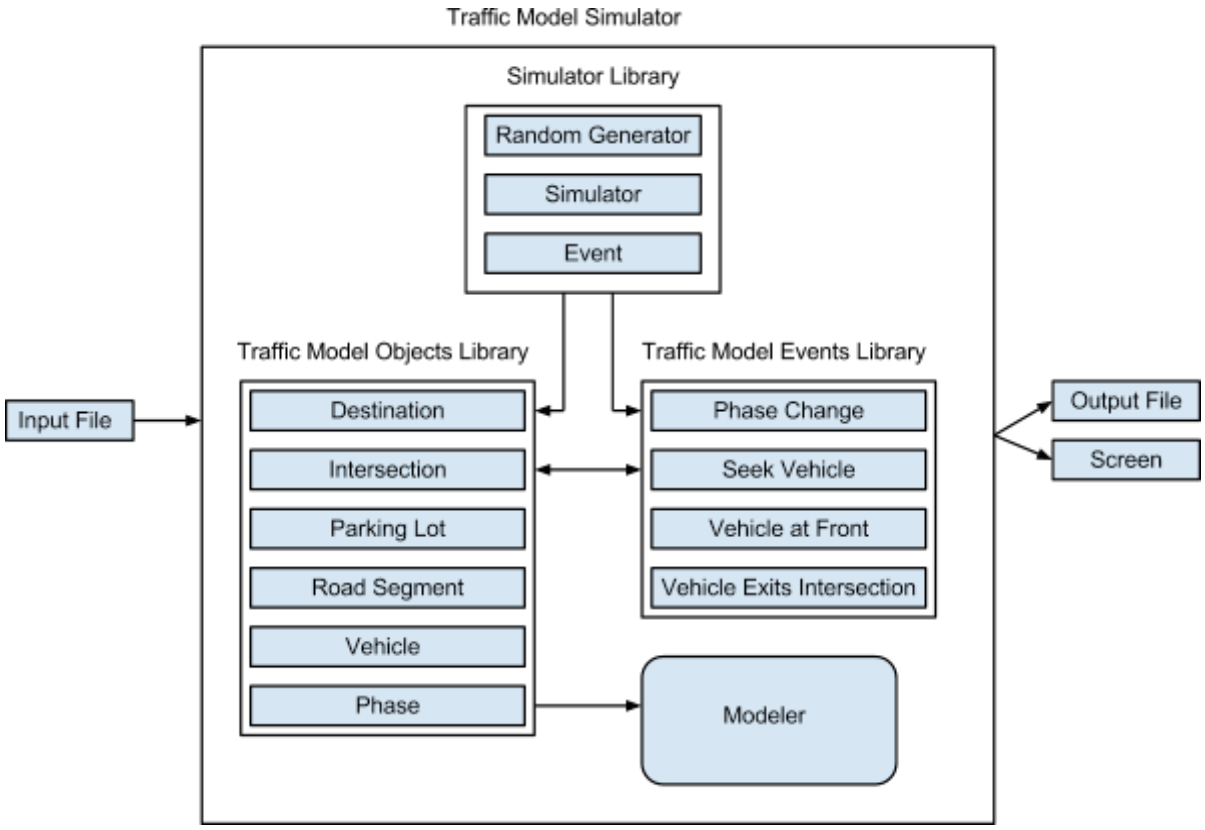


Figure 2: Architectural Diagram of the Traffic Model Simulator

3.1 Modeler

The Simulation Model creates a traffic environment by scanning various parameters (road length, number of intersections, average service time, etc) from an input file. It then uses objects in the Simulator Library to instantiate such an environment; Prior to execution, the model is checked for consistency against the input parameters but also in its data structures (i.e. pointers). Once consistency has been verified, vehicles are instantiated at all parking lots and the simulation kickstarted. After execution of the simulator has completed, the model outputs data (simulation time, number of vehicles served, etc) to an output file for analysis.

3.2 Simulator Library

The Simulator Objects Library provides the classes to perform the simulation of the model. The Simulator Class is the interface for any priority queue simulator. There are two implementations of priority queues in this library. A Linked List and a Calender Queue. A Random Number Generator is also implemented in this library. It also provides the Event Class as an interface for any event used with the simulators.

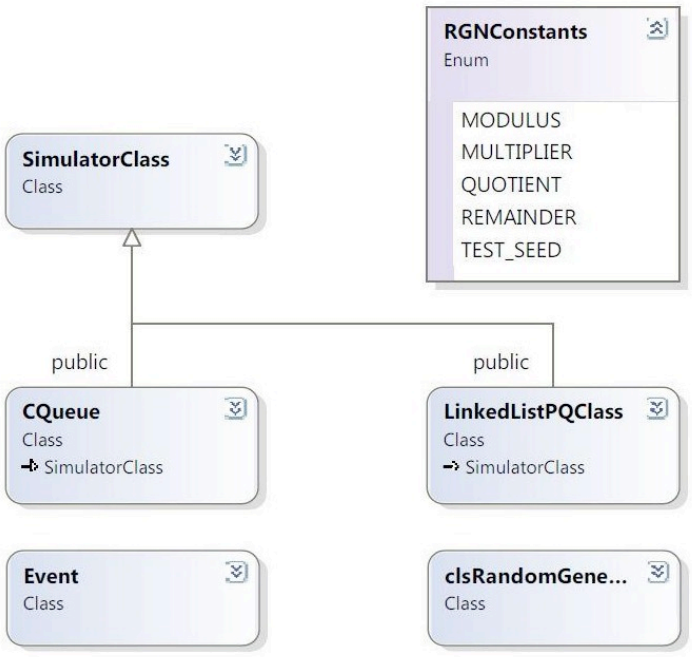


Figure 3: Simulator Objects Library, class hierarchy

3.2.1 Event Class

The Event Class is the basic class for the events to be implemented in the Events Library. There is one property in this class, the EventTime. This class provides public methods to set and get the Event Time, and a public virtual method Run() which must be implemented in the derived classes.

Also added, although not required, the class has a Next property to facilitate the use of linked lists.

3.2.2 Simulator Class

The Simulator Class is the basic interface for priority queues. This class facilitates the use of different implementations of priority queues. In this project for example there were two priority queues implemented; the Calendar Queue and the Linked List.

The public interface of this class provides the following methods:

AddEvent(Event)	Adds an object of class Event to the Queue
PopNext()	Gets the Event with the lowest EventTime in the Queue
PopEvent(Event)	Removes a given Event from the Queue
IsEmpty()	True if there are no Events in the Queue
Reset()	Initializes the Queue

The backend of this event scheduler is the Calendar Queue. The calendar queue is basically an array of pointers pointing to the heads of number of linked lists. Each of these head pointer is called a bucket. A bucket is a linked list which contains all the events scheduled to occur within a certain time frame sorted in ascending order of time. The maximum range of times held in the bucket is called its width and it is common in all the buckets. A pointer is used to track the bucket to dequeue from.

When an event is scheduled with *AddEvent(Event)*, the event is enqueued in the appropriate bucket's linked list. The bucket to which the event is sent to is decided by the event time and is simply calculated by *i=(int)(EventTime/width)%nbuckets* and enqueued in the

appropriate linked list. When a dequeue operation is done with *PopNext()*, the earliest event from the bucket pointed to by the pointer is removed from the linked list. If there is no early event in the bucket, the pointer is incremented and goes down bucket by bucket till the earliest event is found. It would be noted by astute readers that a bucket can hold a non-contiguous set of events since events from more than one range of events(*current range+width*nbuckets*) could be enqueued in the same bucket. Collision is prevented by another counter which records the maximum possible time to which an event could be dequeued from during the current cycle. Thus, even if a bucket is empty, if the earliest event in the bucket exceeds this time, it would not be dequeued. Finally, *PopEvent(Event)* is used to remove an arbitrary event from the bucket so as to deal with phase changes in the intersections which decide which direction cars are allowed to go in. This is done by calculating the bucket where the event is stored and removing that node from the linked list.

Another feature is that the number of buckets is made resizeable so as to reduce the number of empty buckets while maintaining shallow linked lists where not many traversals are required for the enqueue and dequeue operations. Thus, at any point in time, a maximum and a minimum number of events is defined for the calendar. If the number of events goes out of this range, the buckets are resized. This is done by collecting data about the average separation in the time between the first few events in the list and using this data to calculate a new width for the buckets. Then, the number of buckets are reallocated(half or double the original number depending on which direction the number of events broke the range) and the events are enqueued into this new bucket. The range values are kept near powers of two so that resizing operations need not be done often. This would mean the queue which maintains approximately a constant number of events throughout the bulk of the run would not need to resize at all. This reduces execution time tremendously.

3.2.3 Random Number Generator

The clsRandomGenerator provides randomness to the model. Its public interface has two methods:

IntUniform(int, int) Provides a uniform integer random number in the interval given in the argument list.

UniformDist(double, double) Provides a uniform double random number in the interval given in the argument list.

In the simulator we have the random number generator, which is called for determining the vehicle destination issued in the parking lot, this is an integer. The RNG is also called for the vehicle crossing time through an intersection and the amount of time for each phase. We assumed a uniform distribution of values for each call. For the uniform distribution, we give a range of values based on the estimated average time and an assumed error probability of +-5%; the uniform distribution will return a value based on the input and random number. We used the Lehmer algorithm since it is fast and well tested. The algorithm generates pseudorandom numbers meaning it generates a stream of numbers that appear random but can be repeated with the initial seed. We ran our simulations sequentially and passed the new seed for the random number to insure each run did not repeat a stream of pseudorandom numbers.

The Lehmer Algorithm we implemented has four integer and constant parameters; the modulus, the multiplier, the quotient, and the remainder. We used the numbers suggested by Leemis and Park[9], and a remainder that is less than the quotient, which helps to reduce an overflow. The random number generator was tested with the Chi-Squared Test and the results conclude that our random number generator produces independent and identically distributed random numbers that are uniformly distributed. Please refer to the reference manual.

3.3 Traffic Model Objects Library

While implementing the Objects Library, we recognized that Destinations and Intersections shared certain parameters and events. It also happens between Road Segments and Parking Lots. Therefore we decided to create the basic classes Traffic Node, and Vehicle Queue. Then the model was then simplified to a network of Vehicle Queues connected by Traffic Nodes.

TrafficModelObject : This class is the basic class for all objects in the library. Its only public property is Type() which will identify the type of object in the model.

TrafficModelClass : This class is the container of the Model. Basically this class is the Model. Among other properties, the TrafficModelClass has a list of Traffic Nodes, a list of Vehicle Queues, a Simulator, and a Random Generator. It provides methods to create and add objects to these lists to help building the model.

VehicleClass: The most important property of the VehicleCLass is Destination. This property is used at the Nodes to identify the next VehicleQueue in the path.

TrafficNodeClass: This class is the primitive class for DestinationClass and IntersectionClass. This class cannot be created by itself. Among other properties the TrafficNodeClass has a list of Phases, a list of VehicleQueues representing the Road Segments going out of the Node, and the number of Vehicles using the intersection. It provides the method to create and add Phases.

IntersectionClass: The IntersectionClass provides an alternate implementation of certain methods of the TrafficNodeClass that have different behavior between Destinations and Intersections. i.e. When a vehicle enters an Intersection, the Intersection passes it to a Road Segement. The Destination, instead, will collect data and destroy the vehicle object.

DestinationClass: Besides TrafficNodeClass properties inherited, the DestinationClass has properties to store the data collected from the vehicles. It also provide alternative implementations of the TrafficNodeClass methods as described above.

PhaseClass: The properties of the PhaseClass are the Average Service Time, and a list of VehicleQueues using the Phase. It provides methods to get and set its properties.

VehicleQueueClass: This class is the primitive class for ParkingLotClass and RoadSegmentClass. This class cannot be created by itself. Among other properties the VehicleQueueClass has Capacity, number of Lanes, start Node, end Node, Vehicles in, Average

Travel Time, list of Destinations served, and a list of Vehicles Ready. It provides methods to get and set its properties.

RoadSegmentClass: Provides an alternate implementation of certain methods of the VehicleQueueClass that have different behavior between Road Segments and Parking Lots. i.e. When a vehicle exits the VehicleQueue, the RoadSegmentClass passes it from its list of Vehicles Ready. The Parking Lot, instead, creates a new Vehicle to be passed.

ParkingLotClass: Provide alternative implementations of the VehicleQueueClass methods as described above.

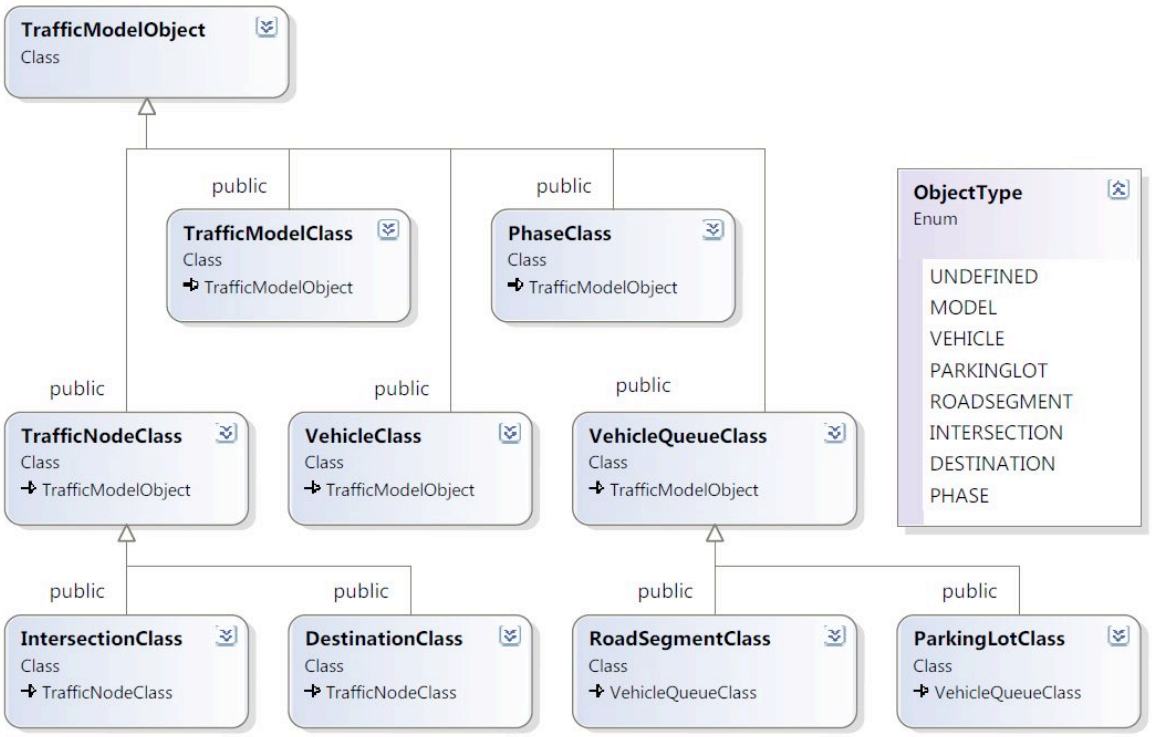


Figure 4: Traffic Model Objects Library, class hierarchy

3.4 Traffic Model Events Library

Combining the required events for Parking Lots and Road Segments, as well as the events of Destinations and Intersections, the required events list in 2.3.2 is reduced to:

Events at Vehicle Queue:

1. **Vehicle Enters Vehicle Queue**
2. **Vehicle is Ready to Exit**
3. **Vehicle Exits Vehicle Queue**

   Events at Traffic Node:

4. **Phase Change**:
5. **Vehicle Enters Intersection**:
6. **Vehicle Exits Intersection**:

Vehicle Exits Vehicle Queue, and Vehicle Enters Intersection happen at the same instant on time. Also, Vehicle Exit Intersection, and Vehicle Enters Vehicle Queue happen at the same instant on time. Therefore, the required number of events is reduced to four.

The **TrafficModelEvent** is an interface derived from Event. The only purpose of this interface is to provide the Type() property to identify the type of event. All four events implemented in this library are derived from this class.

The description for the implementation of the Run() method for each event is as follows:

PhaseChangeEvent: Has a pointer to a TrafficNode

1. Call the Node ChangePhase method. This method will set active the next Phase with Vehicles Ready to cross the Node, and return the Service Time for that Phase.
2. If the Node is Empty (Vehicles In == 0), it will schedule a SeekVehicleEvent per each Line of each Vehicle Queue In in the activated Phase.
3. if the Node is NOT in Idle state, it will schedule the next PhaseChangeEvent, adding the time returned in 1.

SeekVehicleEvent: Has pointers to one TrafficNode and one VehicleQueueIn

1. Run only if the active Phase of the TrafficNode is equal to the Phase serving the VehicleQueue

a. If the VehicleQueue has a VehicleReady, and the VehicleQueue Out serving the Destination of the Vehicle is not full
    i. Call VehicleQueueIn->VehicleOut, and TrafficNode->VehicleIn
    ii. Schedule the next SeekVehicleEvent for the same pair TrafficNode, VehicleQueue adding the random headway time estimated.

VehicleExitsIntersection: Has pointers to a Vehicle, a TrafficNode, a VehicleQueueIn, and a VehicleQueueOut

1. Call TrafficNode->VehicleOut, which return the time the Vehicle needs to travel to the end of the VehicleQueueOut
2. Schedule a VehicleAtFrontEvent for the Vehicle and the VehicleQueueOut adding the time returned in 1.
3. If the Node is Empty (Vehicles In == 0), it will schedule a SeekVehicleEvent per each Line of each Vehicle Queue In in the activated Phase.

VehicleAtFrontEvent: Has pointers to a Vehicle and a VehicleQueue

1. if There are Not VehiclesReady in the VehicleQueue,
    a. if The TrafficNode is Idle,
        i. if the VehicleQueueOut serving the Destination of the Vehicle is not full,
            1. Remove the scheduled PhaseChangeEvent from the simulator
            2. Call TrafficNode->VehicleIn wich will process the Vehicle and schedule a SeekVehicleEvent
            3. return
2. Add the Vehicle to the list of VehiclesReady

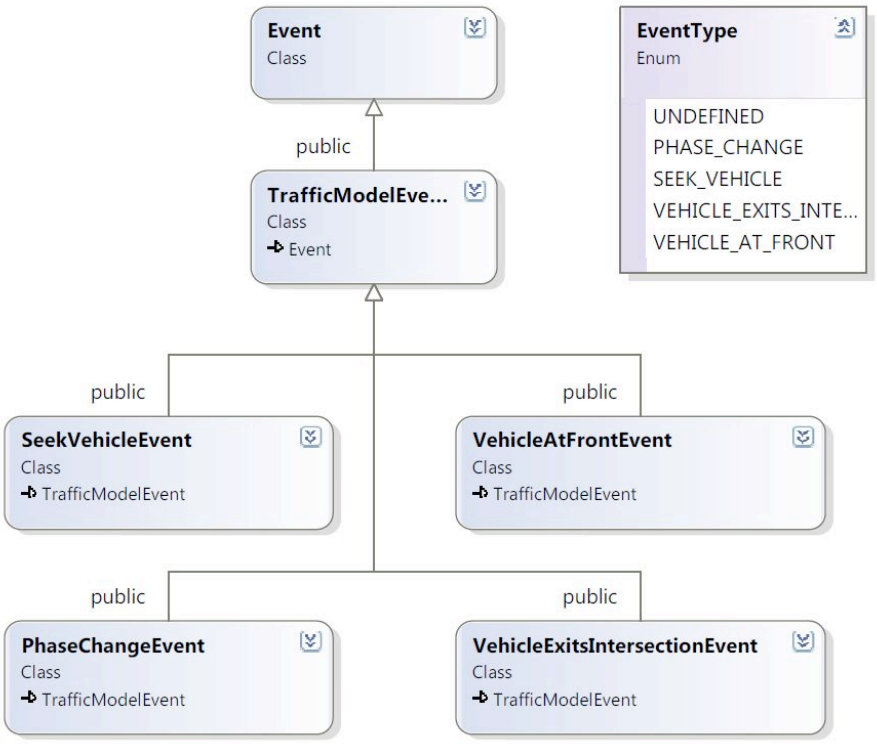


Figure 5: Traffic Model Events Library, class hierarchy

4. Proposed Models

There are 8 parking lots in the Georgia Dome campus with a total capacity of 5468 vehicles. The following table summarizes the information obtained from the Georgia Dome webpage [7].

Parking	Capacity (Veh.)	Lines Out	Location
Red	2000	2	Magnun St.
Orange	580	2	Magnun St & GA Dome Dr.
Silver	82	2	GA Dome Dr.
Blue	722	2	Northside Dr.
Gold	300	1	Northside Dr.
Yellow	1284	2	Northside Dr.

Green	200	3	Marietta St.
Brown	300	2	Magnun St.

Table 1 : Parking Lots and their capacity

4.1 Route 1

The first routing plan was defined by finding the shortest traveling path to the interstate. We used google maps to calculate the shortest route. The plan for Route 1 was to keep it as simple as possible. Therefore, we had one exit per destination and used the fewest number of roads possible. With this design, Northside Drive NW has two way traffic, which would likely cause some congestion. We wanted to minimize the number of road closers or detours for the local traffic. Route 1 was planned to be the simplest model and to cause the least congestion to local traffic not associated with leaving the Georgia Dome.

The first alternative route (see Figure 6 and Table 2 for Route 1 Phases) uses Ivan Allen Boulevard to evacuate the traffic going to Interstate I75/85, and Northside Drive to evacuate the traffic going to interstate I20. The access to I20 West and be through Park Street, and the access to I20 East will be through Oak Street. In the vicinity of the Georgia Dome, Northside Drive will run in both directions, having 2 lines northbound and 2 lines southbound. Marietta Street, and Centennial Olympic Park Drive will be used to conduct the vehicles coming from the Green Lot to either Ivan Allen Boulevard or Northside Drive. Traffic coming from Magnun Street will be directed to Northside Drive through Mitchell Street.

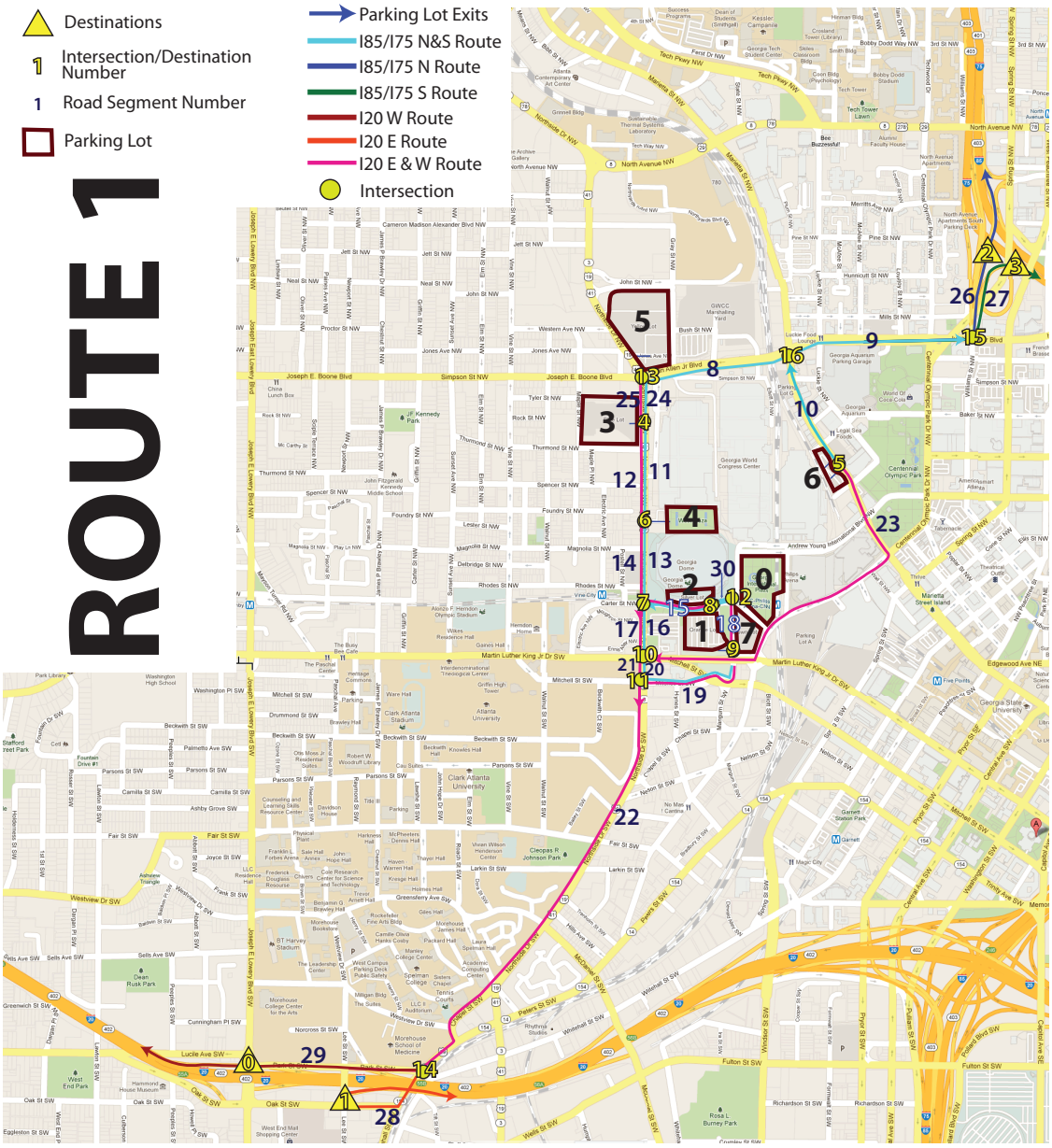
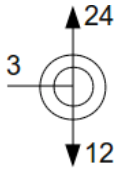
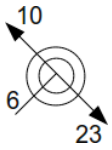
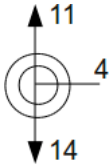
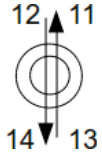
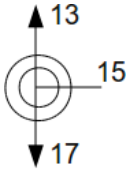
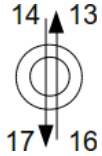
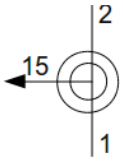
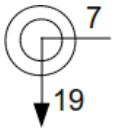
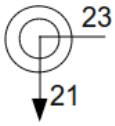
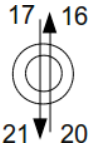
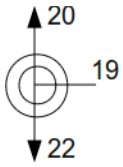
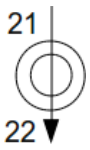
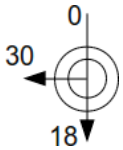


Figure 6: Road map of the routing options for the first route. The base map is taken from Google Maps.

Intersection	Phase 1	Phase 2
4		
5		
6		
7		
8		
9		
10		
11		
12		

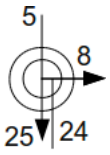
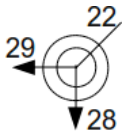
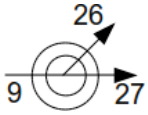
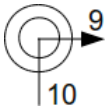
		
		
		
		

Table 2: Proposed Phases for vehicle movement in intersections for Route 1

4.2 Route 2

The second routing plan was designed to have less congestion at the intersections by mapping out multiple routes to the four destinations, two exits for each destination. Route 2 we believed would prove to have the faster travel time and less delay. However, this route would require more resources to close down roads and control traffic. It would also cause greater congestion for the local traffic not associated with the Georgia Dome. Route 2 was also designed to be faster by changing two way roads to one way and therefore using the maximum number of lanes per road section. We also assumed the planned bridge for Mitchel St SW was constructed and in use.

The second alternative route (see Figure 7 and Table 3 for Route 2 Phases) uses Ivan Allen Boulevard to evacuate the traffic from the parking lots located north, going to Interstate I75/85. Martin Luther King Jr Drive and Mitchell Street to evacuate the traffic from the parking lots located south, going to Interstate I75/85. Northside Drive is used to evacuate the traffic going to interstate I20. The access to I20 West and be through McDaniel Street, and the access to I20 East will be through Park Street. In the vicinity of the Georgia Dome, the four lines in Northside Drive will run southbound southbound. Marietta Street, and Centennial Olympic

Park Drive will be used to conduct the vehicles coming from the Green Lot to Luther King Jr Drive. Traffic coming from Magnun St. will be directed to Northside Drive through Mitchell Street.

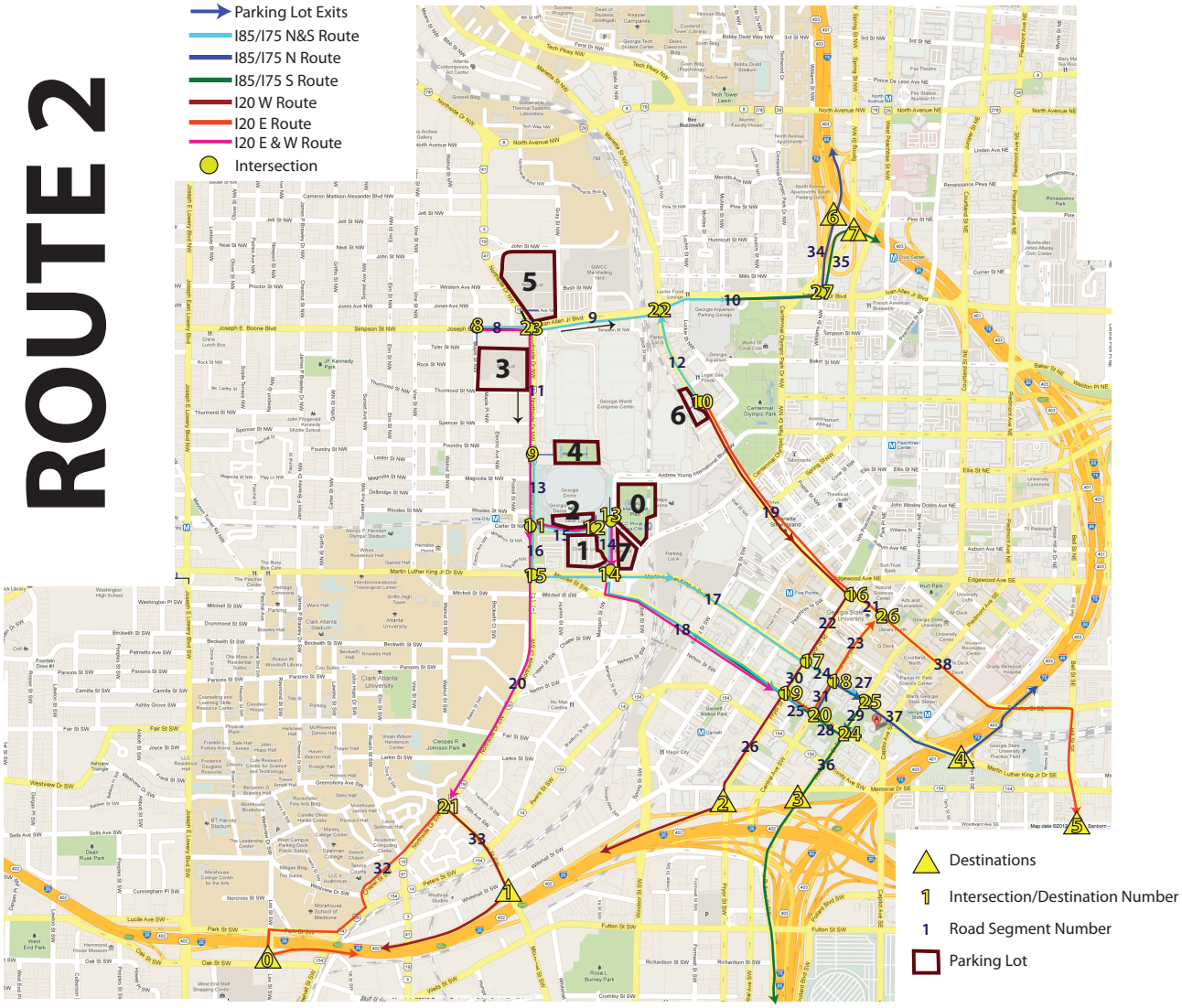
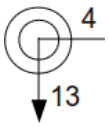
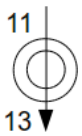
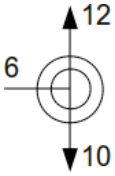
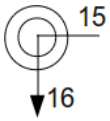
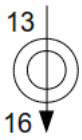
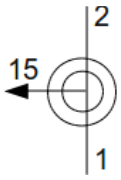
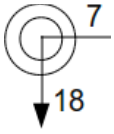
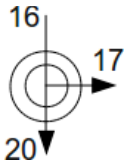
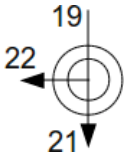
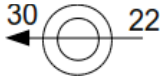
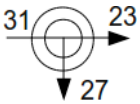
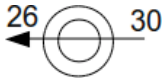
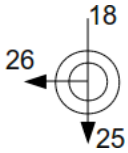
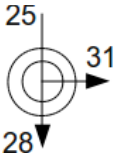
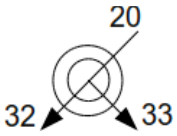
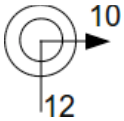


Figure 7: Road map of the routing options for the second route. The base map is taken from Google Maps

Intersection	Phase 1	Phase 2
8		
9		
10		
11		
12		
13		
14		

15		
16		
17		
18		
19		
20		
21		
22		

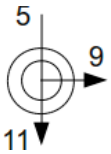
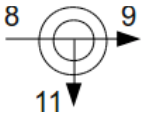
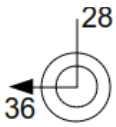
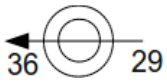
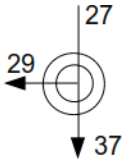
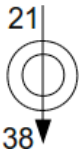
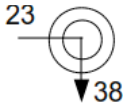
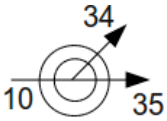
		
		
		
		
		

Table 3: Proposed Phases for vehicle movement in intersections for Route 2

5. Results and Analysis

Each routing plan was run 25,50, 100, and up to 100000 times (each time increased by an order of magnitude). The simulation is able to route vehicles in an order of magnitude less than total vehicles. For example, 1000 vehicles can be simulated in around 100 seconds. Our runs model about 5000 vehicles. Our original goal was to bound the average delay within a confidence of 90%, however the ratio of zeta to the point estimates was already well below a half length size of 10%, thus we tightened our confidence to 99%. Under Routing Plan #1, our goal was for the vehicles to be routed through the shortest path from the parking lots to their

destination to minimize travel time. Both routing plans assume an average vehicle speed on roads of 45 miles per hour. After 1000 runs, we found the average delay to lie within an interval of 20.19 minutes to 20.20 minutes with a confidence of 99%.

ROUTE 1 (99% Confidence)						
ITERATIONS	25	50	100	1000	10000	100000
P-ESTIMATE	1226.624	1220.76	1206.581	1212.072	1212.327	1212.356
STDDEV	26.40208	38.78576	22.8964	40.344	44.46146	44.03874
T	2.796939	2.679952	2.626405	2.58076	2.576321	2.575878
ZETA	2.8743	2.360359	1.25674	0.518367	0.171788	0.054056
CI-MIN (s)	1223.75	1218.399	1205.324	1211.554	1212.155	1212.302
CI-MAX (s)	1229.499	1223.12	1207.838	1212.59	1212.499	1212.41
CI-MIN (m)	20.39583	20.30665	20.08873	20.19256	20.20258	20.20503
CI-MAX (m)	20.49165	20.38533	20.13063	20.20984	20.20831	20.20684

In comparison, Routing Plan #2 aimed to reduce congestion within the heart of the city by routing traffic quickly away and then letting vehicles flow towards their destinations. This too proved to be very effective. After 1000 runs, we can say with a 99% confidence that average delay time lies between 19.07 minutes and 19.08 minutes.

ROUTE 2 (99% Confidence)						
ITERATIONS	25	50	100	1000	10000	100000
P-ESTIMATE	1144.3	1145	1144.6	1144.8	1144.8	1144.8
STDDEV	2.3665	1.9718	2.0489	2.3123	2.3479	2.3634
T	2.7969	2.68	2.6264	2.5808	2.5763	2.5759
ZETA	0.8605	0.5322	0.3759	0.1241	0.0395	0.0125
CI-MIN (s)	1143.4	1144.4	1144.2	1144.7	1144.8	1144.8
CI-MAX (s)	1145.2	1145.5	1145	1144.9	1144.9	1144.8
CI-MIN (m)	19.057	19.074	19.071	19.078	19.08	19.08
CI-MAX (m)	19.086	19.092	19.083	19.082	19.081	19.081

6. Bibliography

[1] Lazowska, Edward D. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Englewood Cliffs, N.J.: Prentice-Hall, 1984.

[2] Discrete event simulation for quick service restaurant traffic analysis Source: Proceedings of Winter Simulation Conference [0-7803-2109-X] Jaynes yr:1994 pg:1061

[3] A cellular automaton model for freeway traffic, Journal de physique. I [1155-4304] Nagel yr:1992 vol:2 iss:12 pg:2221

[4] A fast simulation model for traffic flow on the basis of Boolean operations, Mathematics and computers in simulation [0378-4754] Cremer yr:1986 vol:28 iss:4 pg:297

[5] Law, Averill M, and W. David Kelton. Simulation Modeling And Analysis. 2nd ed. New York: McGraw-Hill, 1991.

[6] Discrete stochastic models for traffic flow, Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics [1063-651X] Schreckenberg yr:1995 vol:51 iss:4 pg:2939

[7] "Campus Parking Map." *Georgia Dome*. Web. 21 Jan. 2012. <http://www.gadome.com/doc/Campus%20Parking%20Map_2010.pdf>.

[8] Randy Brown, Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem, Communications of the ACM, yr: 1998 vol:31 no:10

[9] Douglas W. Jones, An Empirical Comparison of Priority-Queue and Event-Set Implementations, Communications of the ACM, yr: 1986 vol:29 no:4

[9] Lawrence M. Leemis and Stephen K. Park. Discrete-Event Simulation: A First Course. 1st ed. Englewood Cliffs, N.J.: Prentice-Hall, 2006.