Transportation System Simulation

Anita Zakrzewska
Borja Zarco
Marat Dukhan
Yi Wei Cheng

Final Report

1. Introduction

Queuing models have been applied extensively to traffic flow simulations to improve efficiency of many aspects of current traffic infrastructures. *Cheah and Smith* (*1994*) employed a state dependent M/G/C/C queuing model to model the behavior of pedestrian traffic flows. Uninterrupted traffic flow has been modeled with M/M/1 and M/G/1 queuing models by *Heidemann* (*1996*). To include for the effects of congestion, *Jain and Smith* (*1997*) modified the model of *Cheah and Smith* (*1994*) and modeled the vehicular traveling speed as a decreasing function of number of cars in the system. In the recent decade, behavior of non-stationary traffic flow has been modeled as the transient dynamics of M/M/1 queue model by *Heidemann* (*2001*). Many of these early queue models simulate traffic flows with the assumption zero incidents. Recently, *Baykal-Gürsoy et al* (*2009*) explored the impacts of incidents on traffic flows with a steady state M/M/C queuing model.

Priority queue is an abstract data structure that contains a set of elements each with an associated value or priority. The two basic operations on a priority queue are enqueue and dequeue. Enqueue inserts an element into the priority queue while dequeue removes the lowest (or highest) priority element in the queue. In the recent decades, many priority queue implementations have been developed. The simple linear list implementation has complexity of O(n) while the two list method reduces the complexity to O(n0.5). The splay tree (*Sleator and Tarjan*, *1983*; *Tarjan and Sleator*, 1985), pagoda (*Francon et a*l, *1978*), skew heaps (*Sleator and Tarjan*, *1983*; *Sleator and Tarjan*, *1985*), and binomial *(Vuillemin*, *1978*) all have complexity of O(log n). *Brown* (*1988*) developed a calendar queue for the simulation event set problem that is experimentally justified to have complexity of O(1). Recently, *Tang et al* (*2005*) developed a ladder queue that is theoretically justified to have O(1) amortized access time complexity.

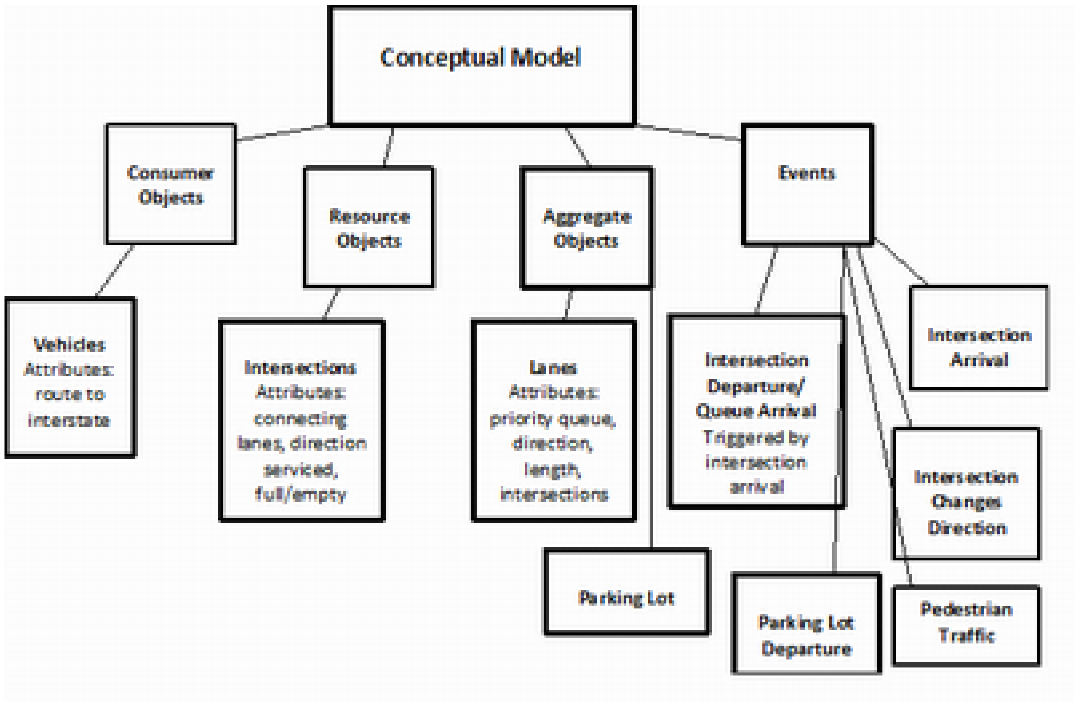
Priority queue models also rely extensively on random number generation for simulation of events. Many random number generators have been proposed (Brent 1994, Matsumoto and Nishimura 1998, Marsaglia 2003) with provably good statistical properties. However, the use of Mersenne Twister (Matsumoto and Nishimura 1998) has become almost standard in numerical simulations: this random number generator is used by default in MATLAB, R, Intel MKL, AMD ACML because it combines good speed with excellent statistical properties. A variety of methods exists for transforming uniform variates into variates from other distributions. One of the simplest methods is inversion of cumulative distribution function, but there calculating inverse CDF may be prohibitively expensive for many distributions. For example, the CDF of Student's t distribution is given by incomplete beta function which is typically calculated by numerical intergration, and multiple evaluations of this function are required to calculate its inverse. However, more efficient sampling techniques were proposed for normal and gamma variates, of which the methods of Marsaglia and Tsang (2000) provide the best combination of speed and statistical accuracy.

In this project, we develop a priority queue model to investigate traffic routing plans that will efficiently route vehicular traffics from parking lots near the Georgia Dome, Atlanta, GA, to the major interstates (I-20, I75/85). The plan should minimize the average delay for a football fan to reach the interstate after the game.

2. Conceptual Model

2.1. Overview

1. The consumer objects in our model are the vehicles passing through the road network. Vehicles contain an attribute that states whether it is a vehicle traveling from a parking lot to the highway or whether it is external traffic. Vehicles traveling from a parking lot to a highway will follow the road network to the specified highway by the shortest path (or shortest path with minimum number of intersections). Through traffic vehicles will follow a random path.
2. The resource objects in our model are the intersections. A finite number of vehicles may occupy an intersection at a time. This number of vehicles depends on the number of lanes in the roads meeting the intersection. For example, if both roads crossing the intersection are two lane roads (one lane per direction), then at most two cars may occupy the intersection at a time.
3. The aggregate objects in our model are the Roads. Each Road has Lane objects and each Lane object contains a list of Vehicles in it.
4. We are modeling only major roads for routing cars from parking lots to highways, but external traffic may be modeled on minor roads.
5. The following events can occur and change the state of the system:
    a. A vehicle emerges from a parking lot (modeled using the random number generator). This is the event that feeds most of the vehicles into the road network being modeled.
    b. An intersection changes the direction it is servicing.
    c. A vehicle leaves a road and enters an intersection. The preconditions for this event are that the intersection must be servicing the direction that the vehicle is traveling and the intersection and road to be entered must not be full.
    d. A vehicle departs from an intersection and enters a new road.
6. Pedestrian crossings at intersections are modeled implicitly. Pedestrians who wish to cross North/South are assumed to cross when the intersection is servicing traffic from its North/South lanes (going straight, not turning).

1. 
Conceptual Model
Consumer Objects
Resource Objects
Aggregate Objects
Events
Vehicles Attributes: route to interstate
Intersections Attributes: connecting lanes, direction serviced, full/empty
Lanes Attributes: priority queue, direction, length, intersections
Intersection Departure/ Queue Arrival Triggered by intersection arrival
Intersection Arrival
Intersection Changes Direction
Parking Lot
Parking Lot Departure
Pedestrian Traffic

2.2. Assumptions

Our model relies on the following assumptions:

1. Once individuals reach the interstate they can travel freely to the final destination. This assumption exists in order to only model the traffic flow from parking lots to interstates. Thus, we do not model any traffic jams that may occur on the interstate, which could stop vehicles from entering an interstate.
2. All drivers initially try to take the shortest route from the parking lot to the interstate. This is a simplification because not all drivers will know the shortest route. However, since drivers will tend to take short routes to their destination, this is a reasonable way of approximating the routes that cars will take.
3. No road accidents.
4. The occupancy of a parking lot is a random variable. Although the parking lots tend to be quite full during game days, they may not be completely filled up. Thus, we model their occupancy with a random variable.
5. External traffic follows a random process. We have no knowledge of the destinations of external traffic. Thus, the directions they proceed at intersection follows a random process. This simplification allows us to avoid creating a specific path for each vehicle in external traffic.
6. All turns are allowed and possible for external traffic on intersections as long as the directions of lanes connecting to an intersection allow for such a turn. This will allow us to more naturally model the flow of external traffic, which may be traveling in any direction (following a random process).

7. Pedestrian traffic is only allowed on pedestrian crossings, which occur at intersections. Pedestrian traffic on controlled pedestrian crossings (at an intersection) is modelled implicitly. Pedestrians are assumed to cross at the same time as vehicles traveling in the same direction are entering and departing from the intersection. Since these events occur concurrently, such pedestrian crossings are modelled implicitly.
8. Each intersection has a person directing the flow of traffic, overriding any stop signs or traffic lights. This assumption allows us to not model stop signs and to have full control over which direction of traffic an intersection is servicing.
9. Vehicles are travelling at a uniform speed.

2.3. Routing Strategies

We investigate the efficiency of two traffic routing plans. In the first plan (Strategy A), the shortest paths between a specific parking lot to the respective major interstate entrances are delineated. Cars leaving this specific parking lot will be directed along the shortest path depending on their destination interstate. The process is repeated for all the other parking lots. The second routing plan (Strategy B) is implemented as a possible improvement to the first strategy and formulated base on the following ideas: (1). To route cars from parking lots so that the drive near other parking lots' exits as little as possible. (2). To route cars from parking lots around the congested areas even if this will result in longer paths. (3). To make some roads (especially near the parking lot exits) one-way to increase the possible bandwidth. Sometimes this requires changing routes which passed this road in opposite direction. Similarly, these paths will be delineated first for all parking lots. The better plan of the two should minimize the average delay for a football fan to reach the interstate after the game. For each plan, the average delay can be evaluated as the total time for all the cars from the parking lots to reach the interstates normalized by the total number of cars.

The traffic routing plans are enforced by police officers stationed at every intersection. As such each intersection and the assigned police officer represents a server in our model. The service time of the server is determined by the strategy adopted by the police officer at the intersection. Each direction is given 30 seconds to cross and then the police switch which cars they let through (for example they let cars going north-south pass, then cars going east-west, then going north-south but making left turns etc). If the intersection becomes idle (no cars for a while), the police stop switching the mode until cars appear in a road and those cars are let through.

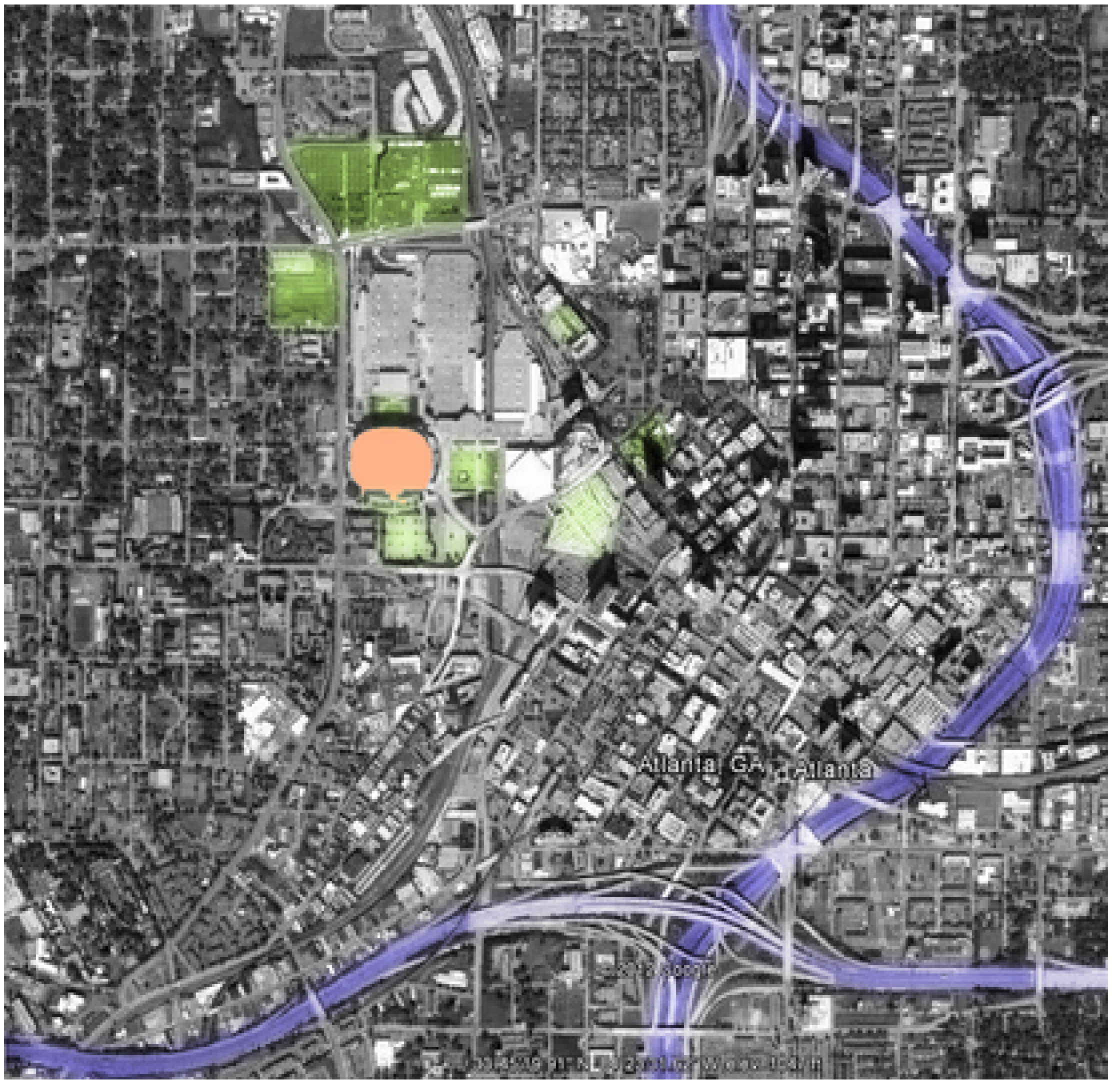
3. Data

Our transportation model relies on information about road network topology and positions of parking lots. The following information is to be collected:

1. Road segments
    a. Traffic direction
    b. Number of lanes in each direction
    c. Length of segment
    d. The two bounding roads
    e. The two intersections on either end
2. Intersections

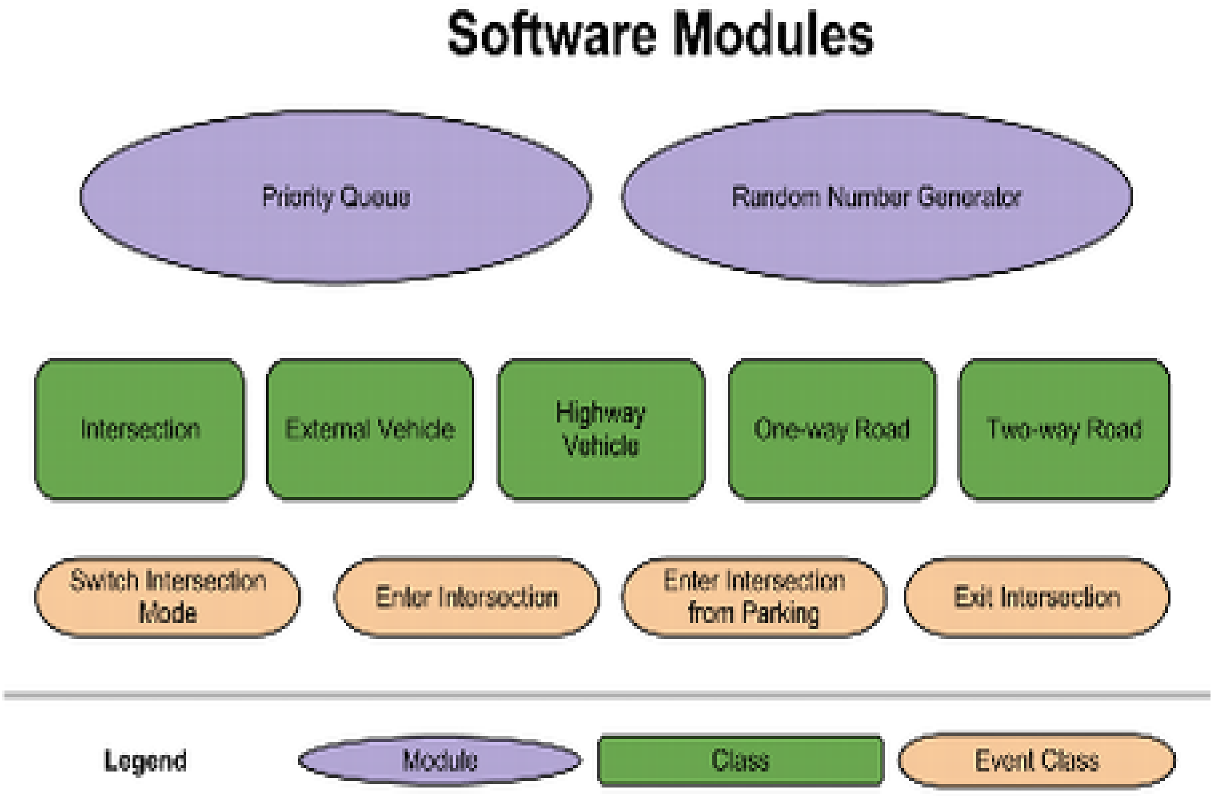
    a. The latitude
    b. the longitude
    c. Whether the intersection is a destination
3. Parking lots
    a. Capacity
    b. The intersection at the exit
4. The shortest paths from each parking lot to each destination

We collected the above information mostly from online maps[1], and also use information on parking lot from Georgia Dome website[2].



[1] http://toolserver.org/~geohack/geohack.php?pagename=Georgia_Dome&params=33_45_27_N_84_24_3_W_type:landmark_scale:2000
[2] http://www.gadome.com/guest/directions/Parking.aspx

4. Software Modules



1. Priority queue (List, Splay, Calendar, and Ladder implementations)
2. Random number generator based on MT19937 (Matsumoto and Nishimura, 1998) and Marsaglia-Tsang (2000) algorithms for normal and gamma variates.
3. Modeled objects
    a. Intersection. A shared resource which is used for driving between the connected road segments. Pedestrian traffic at intersection is modelled implicitly. There is a public intersection interface that is accessed by all other classes. There is an intersection class extending this interface.
    b. Road. Input/output in an intersection. The road class has an internal lane class. Each road has a list of lanes. Each lane has a priority queue associated with it. A road stores information about its length and direction. Have finite capacity. There is an abstract Road class and two non abstract road classes: for one way roads and for two way roads.
    c. Parking lot. A source of cars which leave it. Each parking lot is connected to an intersection on its exit. Parking lots have finite capacity and random occupancy.
    d. Path. This is a list of roads from a parking lot to a destination. Each vehicle has a path.
    e. Vehicle. Vehicles are removed from Parking Lots into the adjacent Intersection. Each non-external Vehicle has a destination intersection and a path of roads leading from the Parking Lot to the destination. Vehicles move along the Roads

(through Intersections) in the path.

4. Modeled events
    a. Intersection arrival. New Vehicle arrives at Intersection from Road. Remove the vehicle from its current road and create and enqueue an Intersection departure event. If the Intersection is a destination, then this is also the Interstate arrival event.
    b. Intersection departure. Vehicle leaves Intersection and is added to a lane in the next road of its path. Notifies the exiting road of the departure, which may cause an intersection arrival event to be enqueued (if there are available vehicles in that road and the intersection mode matches).
    c. Intersection direction change. Switch the serving direction of the Intersection. After the direction of the Intersection is changed, the roads of the intersection are notified. This causes each road to check if it has any vehicles available at that time that can enter the Intersection in its current mode (if the Vehicle's current and next road are allowed in the current mode of the Intersection). If the Intersection is not idle, another Intersection direction change event is created and enqueued.
    d. Parking lot departure. Vehicle leaves Parking Lot and enters the connecting Intersection and then the next Road in its path. Dequeue Parking Lot and determine Vehicle's destination interstate (randomly) and path and set the Vehicle's starting time. Once the Vehicle is added to its first road, if the mode of the intersection still allows for Vehicles to leave the Parking Lot, another Parking Lot departure event is created and enqueued.
    e. Pedestrian traffic. Used at Intersection, Vehicles wait while pedestrians cross the road. Pedestrians cross when the Intersection mode matches (Vehicles are travelling in the same direction as pedestrians), so this is modelled implicitly by Intersection direction changes.

4.1. Priority Queues

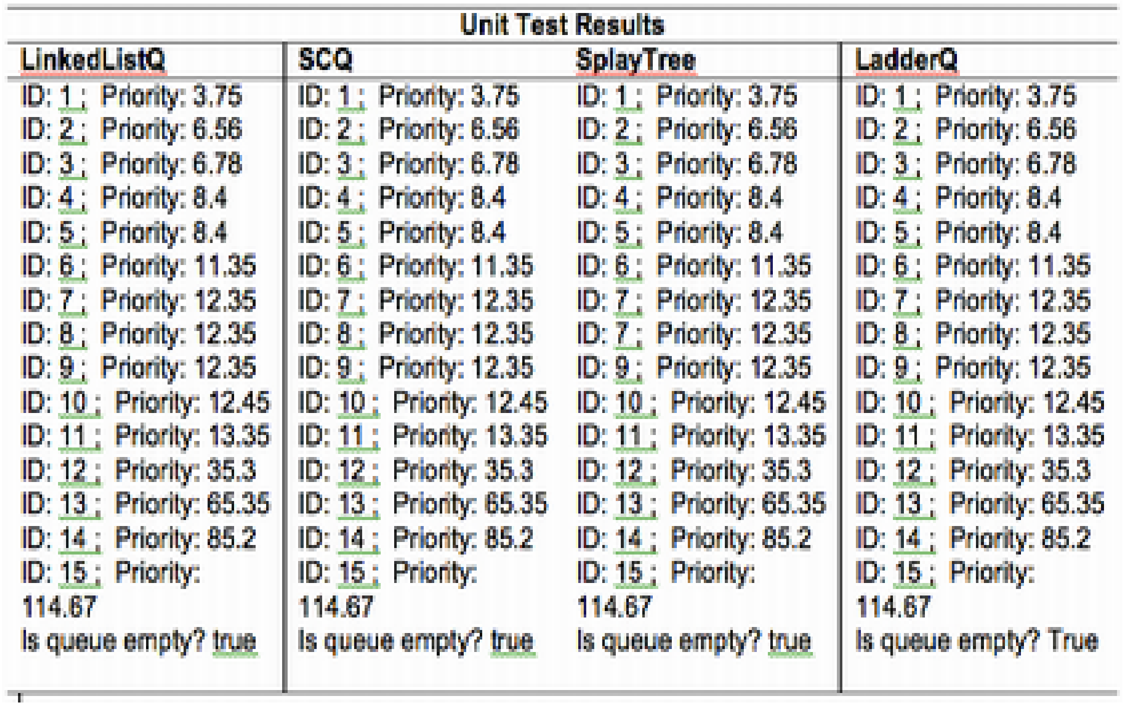
We perform unit tests and performance tests on four known priority queue structures: linked list, splay tree (Sleator and Tarjan, 1985), sorted-discipline calendar queue (Brown, 1988) and the ladder queue (Tang et al, 2005). We first give a brief introduction on the respective queues (except the linked list) in Part 1. In Parts 2 and 3, we present the unit test and performance test results respectively.

The splay tree is a form of binary search tree with the capacity to "self-adjust" data structure during each operation. The self-adjustment/splay operation follows a simple restructuring rule aiming at improving the efficiency of future operations. Theoretically the splay tree have an amortized time bound of O(log n) per hold operation. As such, the splay tree is as efficient as many existing balanced trees such as B-trees and height-balance trees. However, unlike balanced trees, the splay tree is able to maintain its efficiency even for long access sequences, due to the splay operation. During each splay operation, the element that is splayed is placed at the root of tree after restructuring the tree (i.e. tree rotation). As the splay operation is performed immediately after each operation (i.e. enqueue, dequeue), the most recently accessed element is always at the root of the tree, reducing the overheads incurred during searches.

The Sorted-discipline Calendar Queue (SCQ) and the ladder queue are hashing schemes. The SCQ was proposed by Brown in 1988. Heuristically, the data structure is similar to events in a desktop calendar. Arrays of “buckets” are defined, each bucket represents a single day in a year and contains events scheduled in that particular day. Events within the bucket are sorted according to priority (e.g. time). The SCQ achieves O(1) performance when number of elements in the queue, N, and mean timestamp increment, μ, are constant (Ronngren and Ayani, 1997). However, when μ and N vary, O(1) performance is no longer achieved. For example, when μ is varied while N remains constant, only O(N) performance can be achieved. The lost in performance has been attributed to the ineffectiveness of the SCQ resize trigger.

The ladder queue is developed by Tang et al (2005) to overcome the problems experienced by SCQ. The ladder queue consist of three basic parts: (1) Top; which contains an unsorted linked list, (2) Ladder; which contains serveral rungs of buckets, each containing an unsorted linked list, and (3) bottom; which contains a sorted linked list. To overcome the problems encountered by SCQ and maintain O(1) performance the ladder queue follow four principles: (1) Sorting only occurs when high priority events are close to being dequeued, (2) During enqueue operations, only when the Bottom structure becomes too populated, then does a resize operation take place, (3) Bucketwidth refinement during dequeue operations occur on a bucket-by-bucket basis, and (4) Ladder queue generates optimal operating parameters dynamically in accordance to events in its structure.

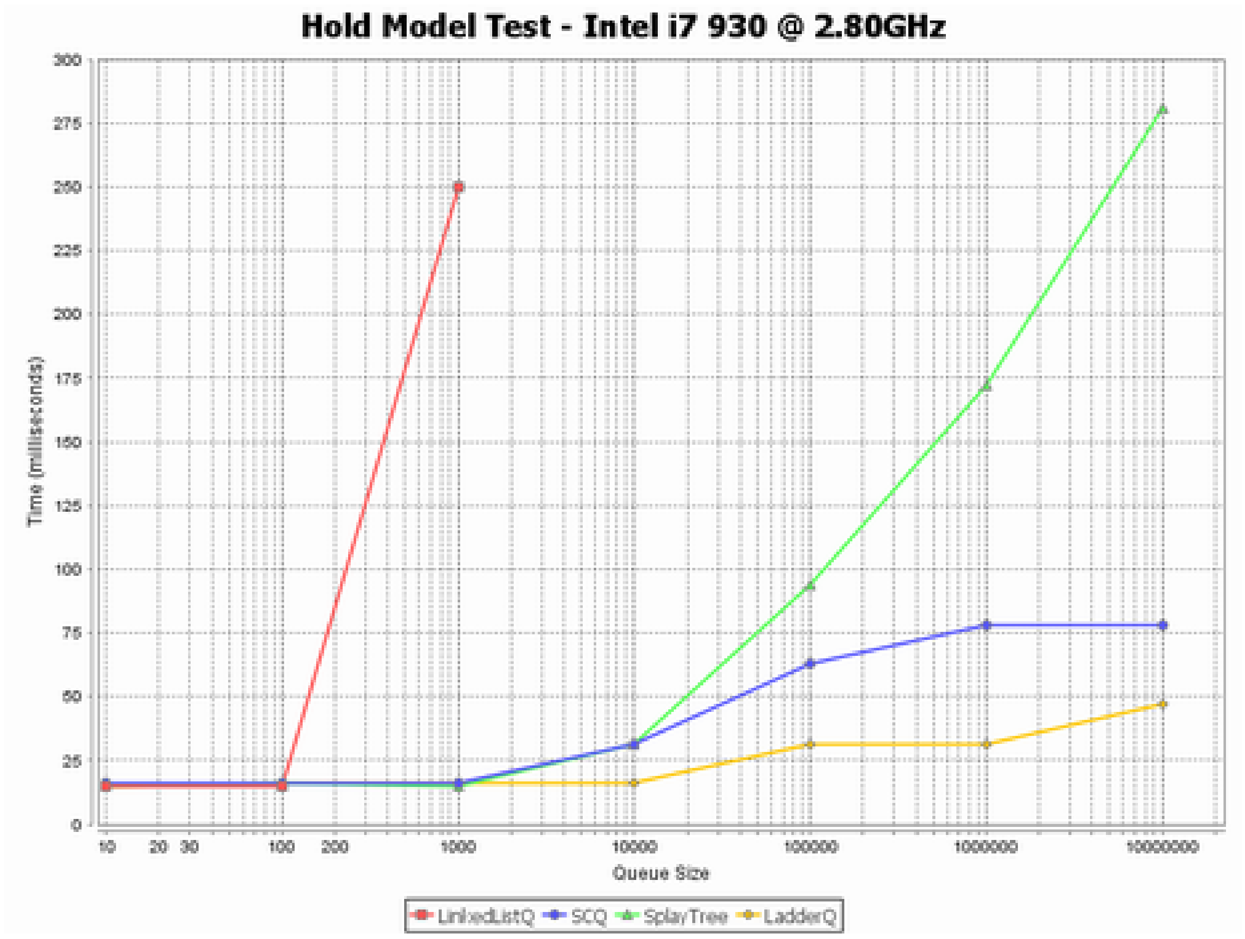
We utilize a model that consist of enqueue and dequeue operations to test the priority queues that we have coded. In this model, fifteen events are initialized with random priorities and indices. First, these fifteen events are enqueue into the priority queue under testing. Next, the events with the smallest priorities are sequentially dequeued until the priority queue becomes empty. If the priority queue is functioning correctly, the result will be a sorted sequence of events from low priority to high priority.



Unit Test Results			
LinkedListQ	**SCQ**	**SplayTree**	**LadderQ**
ID: 1 ; Priority: 3.75	ID: 1 ; Priority: 3.75	ID: 1 ; Priority: 3.75	ID: 1 ; Priority: 3.75
ID: 2 ; Priority: 6.56	ID: 2 ; Priority: 6.56	ID: 2 ; Priority: 6.56	ID: 2 ; Priority: 6.56
ID: 3 ; Priority: 6.78	ID: 3 ; Priority: 6.78	ID: 3 ; Priority: 6.78	ID: 3 ; Priority: 6.78
ID: 4 ; Priority: 8.4	ID: 4 ; Priority: 8.4	ID: 4 ; Priority: 8.4	ID: 4 ; Priority: 8.4
ID: 5 ; Priority: 8.4	ID: 5 ; Priority: 8.4	ID: 5 ; Priority: 8.4	ID: 5 ; Priority: 8.4
ID: 6 ; Priority: 11.35	ID: 6 ; Priority: 11.35	ID: 6 ; Priority: 11.35	ID: 6 ; Priority: 11.35
ID: 7 ; Priority: 12.35	ID: 7 ; Priority: 12.35	ID: 7 ; Priority: 12.35	ID: 7 ; Priority: 12.35
ID: 8 ; Priority: 12.35	ID: 8 ; Priority: 12.35	ID: 7 ; Priority: 12.35	ID: 8 ; Priority: 12.35
ID: 9 ; Priority: 12.35	ID: 9 ; Priority: 12.35	ID: 9 ; Priority: 12.35	ID: 9 ; Priority: 12.35
ID: 10 ; Priority: 12.45	ID: 10 ; Priority: 12.45	ID: 10 ; Priority: 12.45	ID: 10 ; Priority: 12.45
ID: 11 ; Priority: 13.35	ID: 11 ; Priority: 13.35	ID: 11 ; Priority: 13.35	ID: 11 ; Priority: 13.35
ID: 12 ; Priority: 35.3	ID: 12 ; Priority: 35.3	ID: 12 ; Priority: 35.3	ID: 12 ; Priority: 35.3
ID: 13 ; Priority: 65.35	ID: 13 ; Priority: 65.35	ID: 13 ; Priority: 65.35	ID: 13 ; Priority: 65.35
ID: 14 ; Priority: 85.2	ID: 14 ; Priority: 85.2	ID: 14 ; Priority: 85.2	ID: 14 ; Priority: 85.2
ID: 15 ; Priority: 114.67	ID: 15 ; Priority: 114.67	ID: 15 ; Priority: 114.67	ID: 15 ; Priority: 114.67
Is queue empty? true	Is queue empty? true	Is queue empty? true	Is queue empty? True

Results show that both the SCQ and the splay trees are able to perform the enqueue and dequeue operations well.

We utilize the hold model to measure the performance of four priority-queues: linkedlist, SCQ, splaytree and LadderQ. In the hold model, the enqueue operation schedules events into the priority queue. The dequeue operation searches for the event with the lowest/highest priority and removing it from the priority queue. In one iteration, the hold model dequeues an event, randomly alters the priority of the event and enqueue the event back into the priority queue. This process is iterated for N times, where N is the size of the priority queue. At the end of the simulation, average time for the operation is evaluted by normalizing total simulation time by N. We evaluate the average operation time for queue sizes ranging from 10 to 10000000 to obtain performance curve for each priority queue. We then compare the performance of the four priorities by plotting all the performance curve together.



As expected results show that the Linkedlist queue has the worst performance. Performance time of the Linkedlist queue increases exponentially with size, and quickly goes beyond 300 ms when queue size becomes greater than 1000. As expected, the LadderQ has the best performance. This is consistent with results from Tang et al (2005), which showed that the LadderQ has better performance than both the splaytree and SCQ. Performance between splaytree and calendar queue is similar for queue sizes less than 10000. However, beyond 10000, simulation time of the splaytree increases at a much faster rate than the SCQ. At queue size 10000000, the performance time of the splaytree is >3X that of the SCQ and ~7X that of the LadderQ.

Base on the unit test and the performance results, we choose the the LadderQ as the main form of priority queue that we will implement in our traffic simulation model.

4.2. Random Number Generator

In our random number generator we generate implemente Mersenne Twiste 19937 algorithm by Matsumoto and Nishimura (1998) for generating integers. Random double precision floating-point values are obtained by dividing random integers by their maximum value. We also implemented sampling from non-trivial distributions using three basic algorithm: Marsaglia and

Tsang (2000) ziggurat method for generating random variables, Marsaglia and Tsang (2000) acceptance-rejectance method for generating gamma variates, and the method from “Numerical Recipes” by Press, Teukolsky, Vetterling, and Flannery for generating Poisson variates. Using these three distributions and continuous uniform distribution we generate exponential variates as a log-transform of continuous uniform variates, log-normal variates as an exponent of normal variates, Chi-squared variates as a special case of gamma variates, and Student-t variates as a function of normal and Chi-squared variates.

We implemented a number of unit tests to validate the support of random variates. In this tests we generate one million random variates and check that
1. Continuous uniform [0, 1) variates always fall into [0, 1) range.
2. Continuous uniform [a, b) variates always fall into [a, b) range.
3. Log-normal variates are never less than zero.
4. Gamma variates are never less than zero. This test is performed for three different values of parameter α because different algorithms are used depending on this parameter (one algorithm for $0<\alpha<1$, another for $\alpha=1$, and the other one for $\alpha>1$)
5. Exponential variates are never less than zero.
6. Chi-squared variates are never less than zero.
7. Poisson variates are never less than zero. This test is performed for two different values of parameter λ because two different algorithms are used depending on whether parameter λ is more than 5 or not.
All the above tests are successfully passed for our random number generator.

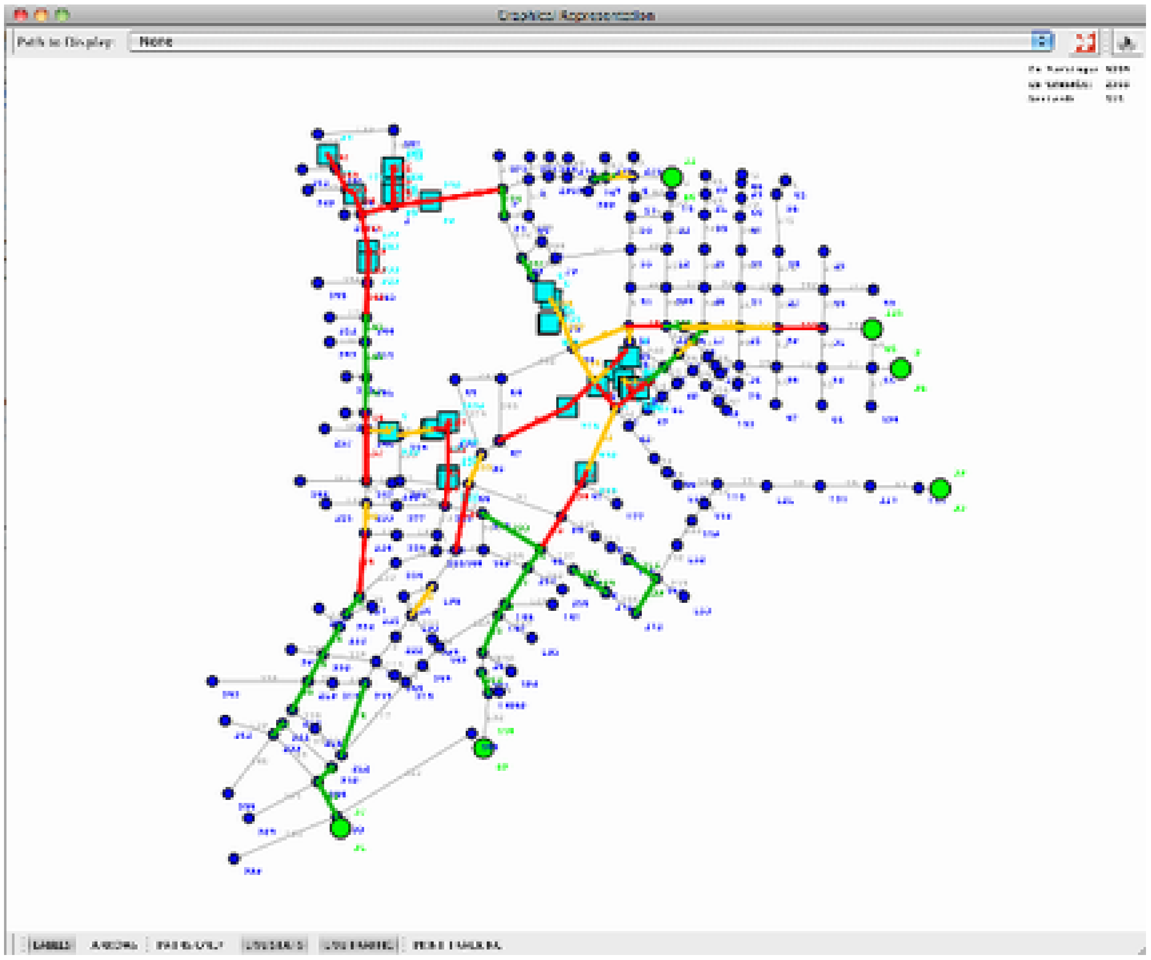
We employ Pearson’s Chi-squared test to validate the statistical properties of variates generated with our random number generator. We do this statistical testing for discrete uniform (random integer), continuous uniform on [0, 1), standard normal, exponential ($\mu = 1$), gamma ($\alpha = 0.5$ and $\beta = 1$; $\alpha = 1.0$ and $\beta = 1$; $\alpha = 1.5$ and $\beta = 1$) variates by dividing the support of the respective probability distributions into 1024 bins so that a random variates falls into each bin with equal probability. Then we generate on average 1000 random variates per bin, and compute the Chi-squared statistics. If the value of the statistics is overly large (above 99.9% quantile of Chi-squared distribution with 1023 degrees of freedom) the hypothesis that the variates are generated from the necessary distribution is rejected.
We also perform Chi-squared test for Poisson distribution with $\lambda = 4.9$ and $\lambda = 5.1$ by dividing the (discrete) support into bins X=0, X=1, …, X=14, X≥15, generating (on average) 1000 samples per bin, and computing the Chi-squared statistics. The critical value in this case is the 99.9% quantile of Chi-squared distribution with 15 degrees of freedom.
The Pearson’s Chi-squared test did not reject our random variates except for the Poisson variates with $\lambda = 5.1$. For this latter variates the Chi-squared statistics is consistently high which suggests that Poisson variates are not of high statistical quality for this parameter value, albeit we borrowed it from a well-renowned book.

To evaluate the performance of our random number generator implementation we measure the time to generate 1000000 variates. The performance numbers (in millions of samples per second) for the Intel Core i7 2630QM CPU with Java 1.6.0.27 for Windows x64 are shown in the table below.

Distiribution	**Sampling Performance (millions of samples per second)**
Discrete uniform (random integer)	29.874
Continuous uniform on [0, 1)	41.594
Continuous uniform on [-10.0, 10.0)	42.359
Standard normal (μ = 0, σ = 1)	27.246
Normal (μ = -5.0, σ = 17.0)	28.153
Log-normal (μ = -5.0, σ = 17.0)	9.564
Gamma (α = 0.5, β = 5.0)	9.695
Gamma (α = 1.0, β = 5.0)	16.451
Gamma (α = 1.5, β = 5.0)	14.804
Exponential (μ = 5.0)	17.421
Chi-Squared (v = 42.42)	14.583
Student-t (v = 93.82)	10.042
Poisson (λ = 4.9)	4.514
Poisson (λ = 5.1)	1.818

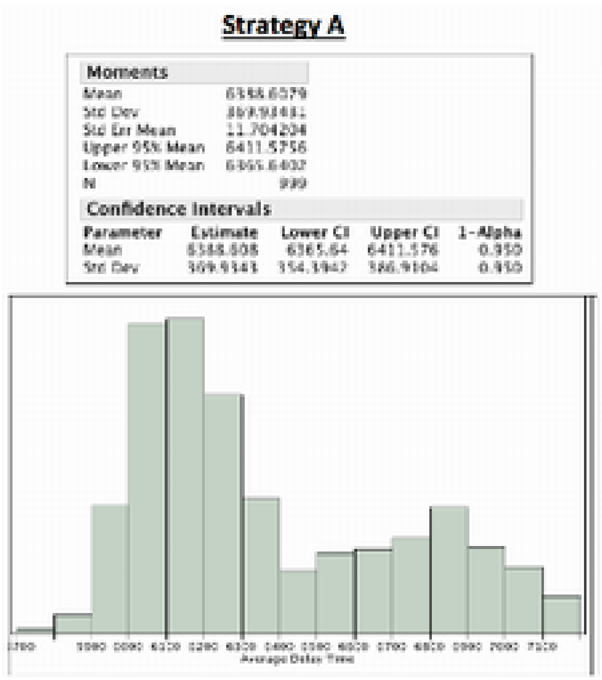
5. Traffic Simulation

We investigate the efficiency of two traffic routing plans. In the first plan (Strategy A), the shortest paths between a specific parking lot to the respective major interstate entrances are delineated. Cars leaving this specific parking lot will be directed along the shortest path depending on their destination interstate. The process is repeated for all the other parking lots. The second routing plan (Strategy B) is implemented as a possible improvement to the first strategy and formulated base on the following ideas: (1). To route cars from parking lots so that the drive near other parking lots' exits as little as possible. (2). To route cars from parking lots around the congested areas even if this will result in longer paths. (3). To make some roads (especially near the parking lot exits) one-way to increase the possible bandwidth. Sometimes this requires changing routes which passed this road in opposite direction. Similarly, these paths will be delineated first for all parking lots. The better plan of the two should minimize the average delay for a football fan to reach the interstate after the game. For each plan, the average delay can be evaluated as the total time for all the cars from the parking lots to reach the interstates normalized by the total number of cars. We also developed a graphic user interface for our model.



For each strategy, we ran the the simulation for 1000 times, reinitializing the parking spaces with different number of vehicles each time. Average delay for all the 1000 runs are populated for each strategy. We then evaluate the statistical properties (i.e. mean, variance, standard deviation, confidence interval) of the set of results for each strategy. Statistics of the results for each strategy allow us to determine which strategy is better within a certain degree of confidence interval. Simulation results show that Strategy A still outperform Strategy B.

Summary of statistics for both strategies. N denotes degrees of freedom.



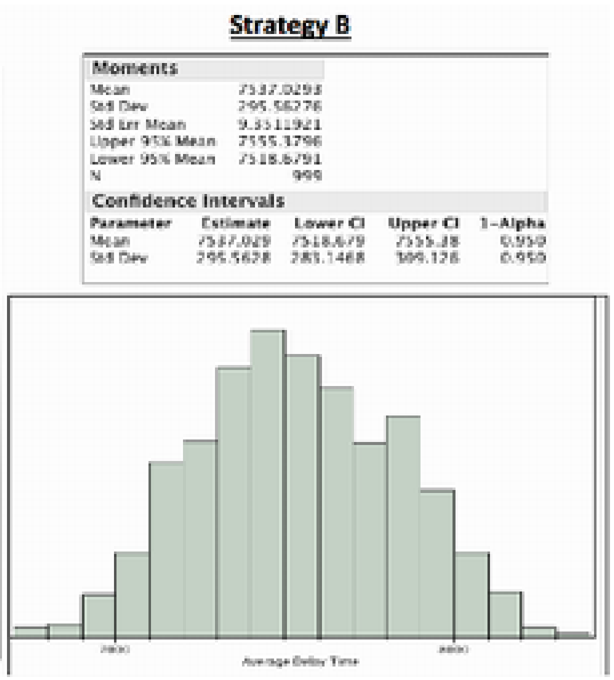
Strategy A

Moments

Mean	6388.6079
Std Dev	369.93481
Std Err Mean	11.704204
Upper 95% Mean	6411.5756
Lower 95% Mean	6365.6402
N	999

Confidence Intervals

Parameter	Estimate	Lower CI	Upper CI	1-Alpha
Mean	6388.608	6365.64	6411.576	0.950
Std Dev	369.9348	354.3947	386.9104	0.950



Strategy B

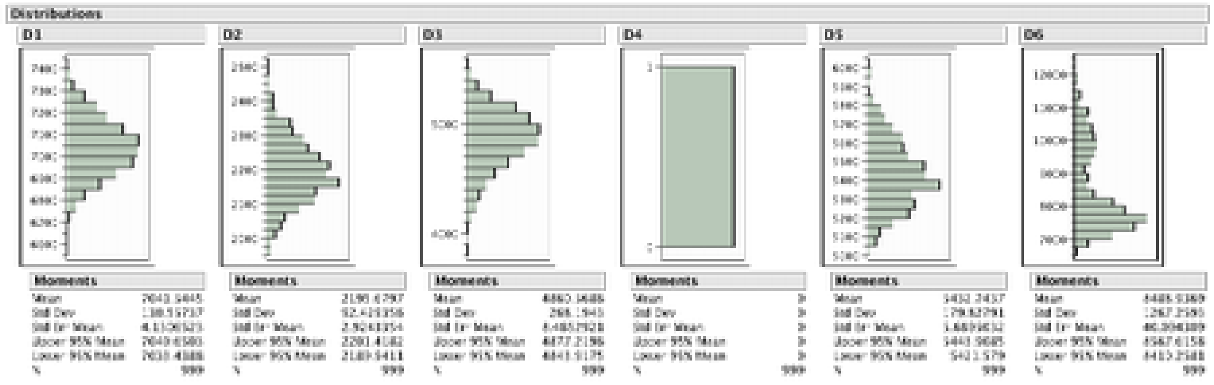
Moments

Mean	7537.0293
Std Dev	295.56276
Std Err Mean	9.3511921
Upper 95% Mean	7555.3796
Lower 95% Mean	7518.6791
N	999

Confidence Intervals

Parameter	Estimate	Lower CI	Upper CI	1-Alpha
Mean	7537.029	7518.679	7555.38	0.950
Std Dev	295.5628	283.1468	309.128	0.950

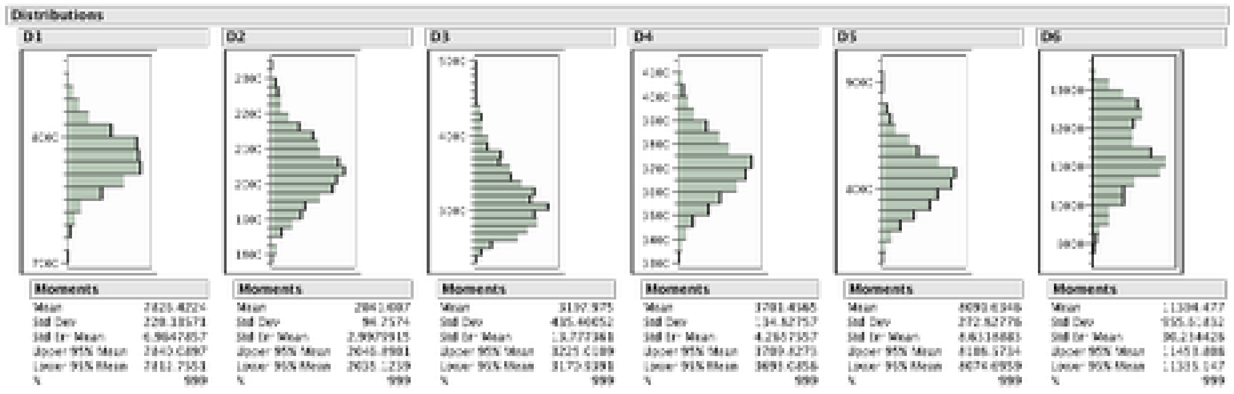
Statistics for each destination for Strategy A. N denotes degrees of freedom.



Distributions

Moments	D1	D2	D3	D4	D5	D6
Mean	7041.5445	2195.6797	4860.3686	0	5432.2437	8488.9369
Std Dev	130.15737	92.419356	268.1945	0	179.82791	1267.2595
Std Err Mean	4.1180525	2.9241054	8.4852928	0	5.6893632	40.094089
Upper 95% Mean	7049.6580	2201.4182	4877.2196	0	5443.3685	8567.6158
Lower 95% Mean	7033.4308	2189.9411	4843.5175	0	5421.1579	8410.2580
N	999	999	999	999	999	999

Statistics for each destination for Strategy B. N denotes degrees of freedom.



Distributions

Moments	D1	D2	D3	D4	D5	D6
Mean	7425.9224	2041.0087	3197.975	3701.4565	8091.6348	11394.477
Std Dev	220.18571	94.7574	435.46052	134.82757	272.82776	955.61832
Std Err Mean	6.9847857	2.9979915	13.777368	4.2657357	8.6318881	30.234428
Upper 95% Mean	7440.0897	2046.8981	3225.0109	3709.8275	8108.5734	11453.808
Lower 95% Mean	7411.7553	2035.1239	3170.9391	3693.0856	8074.6959	11335.147
N	999	999	999	999	999	999

6. Conclusion

We developed a traffic system model that successfully simulate traffic flows along road networks in downtown Atlanta, Georgia. The model is able to route all the vehicles from the parking lots to the highway entrances following both prescribed routing strategies(e.g. no deadlocks, no cars stuck in the road networks). Routing Strategy A surprisingly still outperformed Strategy B, which was prescribed to alleviate the congested intersections in Strategy A. Further analysis revealed that we probably did not route these vehicles far away from the congested areas/intersections. Increasing the path length of some routes and forcing some existing roads to become one way caused longer average delay time in Strategy B.

Here we present the caveat of our model for consideration. It is noted that for both routing strategies, the average delay times are higher than expected: ~1.5 - 2 hrs. We attribute such high delay time to the following factors:
(1). The intersection is behaving like a traffic light. It allows traffic through for a fix amount of time regardless of whether the road has no vehicle.
(2). Cars within the same road do not switch lanes. Therefore, vehicles in a crowded lane will not move to the adjacent lane even if the lane is empty of vehicles.
(3). Further analysis of the bottlenecks revealed that most of the paths have evacuation times of less than an hour. A few paths have evacuation time as long as four hours, which caused the average delay time to be ~1.5hrs. While these paths are the shortest routes from some parking lots to destinations, they by-passed many other parking lots

7. References

Baykal-Gürsoy, M. and Xiao, W. and Ozbay, K. (2009) "Modeling traffic flow interrupted by incidents", European journal of operational research, 195:127–138.
Brent, R.P. (1994) "On the periods of generalized Fibonacci recurrences", Mathematics of Computation, 63:389–402.
BROWN, R. 1988. Calendar queues: A fast *O*(1) priority queue implementation for the simulation event set problem. *Commun. ACM 31*, 10 (Oct.), 1220–1227.
Cheah, J.Y. and Smith J.M., Generalized M/G/C/C state dependent queuing models and pedestrian traffic flows, *Queueing Systems*, 15, 365-385 (1994).
Francon. J., Viennot, G., and Vuillemin, J. Description and analysis of an efficient priority queue representation. In Proceedings of the 19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich.. Oct. 16-18). IEEE, Piscataway, N.J.. 1978, pp. Z-7.
Heidemann, D., A queueing theory approach to speed-flow-density relationships, *Proc. Of the 13th International Symposium on Transportation and Traffic Theory,*France, (July 1996).
Heidemann, D., A queueing theory model of nonstationary traffic flow. *Transportation Science*, Vol. 35, No.4, 405-412 (2001).

Jain, R. and Smith, J. M., Modeling Vehicular traffic flow using M/G/C/C state dependent queueing models, *Transportation Science*, 31, No. 4, 324-336 (1997).
Marsaglia, G., and W. W. Tsang (2000) “The ziggurat method for generating random variables”, Journal of Statistical Software 5:1–7.
Marsaglia, G., and W.W. Tsang (2000) “A simple method for generating gamma variables“, ACM Transactions on Mathematical Software 26:363–372.
Marsaglia, G. (2003) “Xorshift RNGs”, Journal of Statistical Software, 8:1–6.
Matsumoto, M., and T. Nishimura (1998) “Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator”, ACM Transactions on Modeling and Computer Simulation 8:3–30.
Sleator, D.D.. and Tarjan, R.E. Self-adjusting binary trees. In Proceedings of the ACM SIGACT Symposium on Theory of Computing (Boston, Mass., Apr. 25-27). ACM, New York, 1983, pp. 235-245.
Sleator, D.D., and Tarjan. R.E. Self adjusting heaps. SIAM]. Comput.
Tarjan, R.E., and Sleator, D.D. Self-adjusting binary search trees. J. ACM 32, 3 (July 1985). 652-886.
Tang, W. T and Goh, R . S. M and Thng, I. L (2005). Ladder queue: An o(1) priority queue structure for large-scale discrete event simulation. ACM Transactions on Modeling and Computer Simulation. Vol 15, No. 3, 175–204.
Vuillemin, J. A data structure for manipulating priority queues. Commun. ACM 21,4 (Apr. 1978), 309-315.
Zhang, H. and Shi, D. (2010) “Explicit solution for M/M/1 preemptive priority queue”, International Journal of Information and Management Sciences, 21:197–208.