Comparison and Efficiency of Time-Fixed and Adaptive Traffic Signal Control Models*

S. Iyer
NCSSM Online
North Carolina School of Science and Mathematics
Durham, North Carolina

September 3, 2024

Abstract: Traffic congestion and delays are major problems in society, posing not only problems for commuters but also increasing carbon emissions, especially in modern-day cities. To this end, traffic signals are tools through which traffic is organized and efficiently processed to allow commuters to travel safely and quickly. Therefore, traffic signal control models are a crucial area of study that needs to be optimized for each situation to regulate traffic effectively. There are two types of traffic signal methods: time-fixed, where the red and green intervals remain the same, and adaptive, which employs sensors to dynamically adjust intervals to handle real-time traffic changes. To compare these two methods and determine the better usage for these models, we have written a Poisson-generated random arrival traffic simulation in Python based on both models. Then, we ran comparative tests in various traffic conditions to find each model's strengths and weaknesses. Overall, we found the Webster fixed-time model to be more efficient, with the percent difference between the average throughput being 55%. In the future, the author would like to corroborate their results with real data to determine the validity of many simulation assumptions and further research realistic simulation condition parameters to ensure that results correlate to real-world systems.

Key words: Traffic Engineering, Webster's Method, Traffic Signal Control, Computational Transportation Engineering, Traffic Intersection Modeling

*Correspondence to: iyer24s@ncssm.edu

Introduction

Transportation is a very large part of modern society, with more than 290 million cars alone in the United States. This sheer volume of vehicles means that drivers often face delays during their commutes. In 2019, drivers in the United States on average spent more than 54 hours stuck [7] in traffic per year - the equivalent of more than a full work week. Traffic jams are especially bad in urban areas, with cities like Los Angeles or Washington DC having average driver gridlock of over 100 hours. Simply put, traffic jams are significant problems for modern society function.

To this effect, traffic signal lights have been one solution to organize and control the flow of traffic. Well-designed traffic lights allow for efficient flow, decreasing traffic gridlock and, in turn decreasing greenhouse gas emissions, a major contributor to the climate change crisis. In addition, properly organized traffic can yield economic benefits, such as decreasing resource consumption. Put simply, picking and developing the optimal traffic signal control model is crucial to improving traffic conditions.

Traffic signals have three different lights: green, meaning that cars may go through the intersection, yellow, or amber, meaning that cars should slow down in preparation to stop, and red, meaning that cars must be stopped. Together, these signals form a sequence called a *cycle*. The *cycle length* is therefore the amount of time for the completion of a cycle. The cycle length mainly consists of the red and green time intervals, but also includes the change interval, or the amber/yellow time interval, and the all-red time, or the interval where all *phases*, or the paths crossing at an intersection, are red. In many cases, cycle times also account for pedestrians crossing.

In order to optimize traffic light patterns, these interval lengths must be decided to best fit traffic conditions. When collecting traffic data, it is usually important to collect the observed volume or the actual traffic flow, expressed in vehicles per time - in this paper, we refer to this quantity as the volume and refer to the aggregate of idle cars as the queue. It is also helpful to collect the saturation flow or the maximum theoretical vehicular flow.

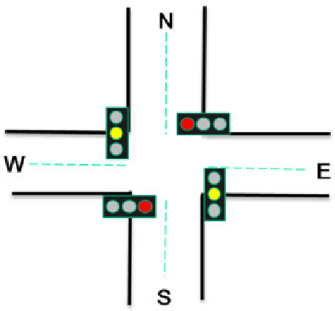

Figure 1: The intersection diagram for this paper.

There are two main categories of traffic signal control models: fixed-time, meaning that all intervals have a preset time limit. These models are used in urban areas, due to their higher volume, regularity, low variance, predictability, and good network organization. An adaptive model, on the other hand, dynamically changes the timings of intervals to equitably distribute green time to all phases. Adaptive models utilize real-time sensor data and usually use machine learning models. In order to choose the best model for a given intersection, it's important to run simulations and determine their efficiencies. Therefore, in this paper, we have written a Python random simulation to compare the throughput (the fraction of cars that pass through an intersection). Figure 1 shows a diagram of the modeled intersection. The intersection here has two phases (two paths that cross); each phase has one lane. The two models used in this paper are Webster's method, a com-

monly used formula-based fixed-time method developed in the 1960s by F.V. Webster through large amounts of data collection and statistical analysis of various intersection geometries and cycle lengths. Additionally, the method assumes random arrivals, which we also assume in this paper through the use of Poisson generation. The other model used in this study is a simple forecast-based adaptive model, which determines the green and red intervals by accounting for current intersection conditions and future expectations.

The object of this study is to determine the better model quantifiably, through comparing throughput, and qualitatively, through making observations on the strengths and weaknesses of each model and their performances in drastically different situations.

Computational Approach

In order to compare the throughput and queue delays of fixed-time and adaptive signal methods, we created a random simulation in Python. Before writing the simulation, however, the methods needed to be described mathematically. Then, the general simulation method and its simplifications also needed to be delineated.

As explained in the Introduction, we considered both a fixed-time and adaptive model. The two most common fixed-time methods presented by the US DOT's traffic signal timing handbook [3] were Webster's method, a formula for calculating the optimal cycle length and its green intervals, and the Greenshields-Poisson Method, a statistical method for determining the Phase Time. Due to its explicit simplicity and ease of calculation, we decided on Webster's method as our fixed-time signal method. Below is the equation for the relevant parameters:

$$C_o = \frac{(1.5 * L + 5)}{(1 - y)} \tag{1}$$

$$G_a = \frac{y_a}{y} * (C_o - L) \tag{2}$$

$$G_b = \frac{y_b}{y} * (C_o - L) \tag{3}$$

Equation 1 calculates C_o, the optimal cycle length, the time it takes to complete one entire sequence of signal phases (ie. Red to Green to Yellow and Red once again). It is dependent on L, the total lost time at an intersection, which is caused by both all-red time (R_a), when all phases are red (in this study, we did not consider the effects of all-red time on throughput), and lost time at a single phase (L_p), due to driver reaction times. The equation for lost time is below:

$$L = (n * L_p) + R_a \tag{4}$$

Above, n represents the number of phases, the number of paths that converge at an intersection. As is standard practice, lost time at a phase is assumed to be 2 seconds [2].

Another important parameter to calculate the optimal cycle time is y, the sum of all critical flow rations. It is calculated as follows:

$$y = \sum_{i=1}^{n} V_i / SV_i \tag{5}$$

Here, V is the actual observed traffic volume for a given phase (given in vehicles per hour) and SV is the saturation flow volume, essentially the theoretical maximum volume assuming a perfect uniform flow of cars. Since this value usually requires large amounts of data collection

over long periods, we have assumed the saturation flow to be 1900 vehicles per hour, a standard value accepted by the Transportation Manual by Oregon State University, Portland State University, and the University of Idaho [4].

From these parameters, we can also find G_a and G_b, the corresponding green phase times for phases A and B (and so on, for n phases). In equations 2, 3, $y_{a,b}$ represent the critical flows for those phases.

Now that Webster's method was chosen for our time-fixed method, we needed to find an adaptive method. Unfortunately, this proved difficult, as much of the literature and practical adaptive methods were developed via machine learning and reinforcement learning from large amounts of data, something we lacked both the tools and time for. Therefore, we developed our own simple adaptive method based on linear forecasting.

The model works as follows: at any given time t, the program takes in the total vehicles in the queue, the rate of exit, the phase vehicle arrival rate, and the phase signals. It forecasts the change in the phase totals through the following equations for each time step. Note that $Q_{g,r}$ are the current queues, $V_{g,r}$ are the traffic rates, and I is the intersection exit rate (cars per second), in this case, 0.2 cars per second.

$$P_g = Q_g + V_g - I$$

$$P_r = Q_r + V_r$$

Figure 2: The green and red phase forecast formulas, applied per each time step. The forecast function returns the switch time - that is, when the queue in P_r is greater than the queue in P_g.

The program continues the forecasting in a while loop while incrementing the time variable until the queue time for the red phase is greater than the green phase. This time of "diminishing returns," as we refer to it, is the switch time. The adaptive simulation continues to run the forecasting function at every time step, and the switch time is updated to reflect current traffic queues. When the switch time is finally reached, the light switches color and the new green phase begins. The switch function continues running to determine the next switch time, and so on and so forth. We believed our adaptive method had merit because it prioritized allowing as many cars through the intersection, regardless of the origin phase, and distributed green time equitably.

A crucial aspect of the simulation is its random nature. Rather than assuming a constant stream of cars coming into an intersection, which would make our results less realistic, we employed a random generation method based on the Poisson distribution, which, being a discrete distribution, seemed perfect for car generation. This was accomplished by using the np.random.poisson() function from the NumPy library.

With the mechanics of our models defined, we looked for any simplifications. In order to yield the most useful and insightful results, while minimizing complexity, we made the following assumptions:

Assumptions:

1. Cars are treated as homogeneous particles in this simulation, rather than having size or any distinct features.
2. Drivers drive into and out of the intersection at a constant velocity, meaning that their rate of exit is uniform.
3. Yellow lights are not included in this simulation, since they effectively serve as green lights. Only red and green are.
4. Turning right is not allowed on a red light, which if allowed, would thereby decrease the queue.
5. Turning Left/Right is assumed to be identical to continuing straight in terms of queue time. In other words, crossing the intersection requires the same time, regardless of driver behavior.

With both our methods mathematically and logically expressed, we turned to programming in Google Colaboratory to create our simulation (the code appendix with documentation is at the end).

Results and Discussion

With our two simulation functions written and any other necessary programming tasks completed, we ran a comparative simulation between the two methods. As outlined in the introduction, we created a simulation with 2 phases, here referred to as the A and B phases. In addition, we ran our simulation for 1 hour, or 3600 seconds, to simulate total traffic over 1 hour. This is also a realistic choice, as traffic volumes and rates often change drastically throughout the day. Our time step, to allow for timely results without too much computational power, was 1 second, meaning that each simulation ran for 3600 time steps. Any time step finer than 1 second wouldn't make a large difference in the results due to the discrete nature of our study. We then chose our intersection cross time to be 5 seconds. As discussed above, this simulation assumes a constant exit speed and uniform flow out of the intersection.

Our collected results were the throughput, calculated as the total number of cars passing through the intersection from one phase divided by the total number of cars arriving at that phase; in other words, we wanted to know the percentage of cars that exit the intersection from a given phase. We also collected the average queue delay, the average time a car has to wait before crossing the intersection.

As an example, Figure 4 shows the car queues over time. The "car stock" curve shows the cars queued at the intersection, and the "cars through" curve shows the total number of cars passing through the intersection since the beginning time. In this case, the car stock remains close to zero for the entire simulation time, which suggests that the fixed-time method is effectively able to process cars at rates equal to or faster than car arrivals.

We ran a comparative simulation with traffic volumes ranging from 100 to 900 cars per hour for both phases A and B, resulting in 81 different comparisons (our interval was 100). To minimize variability in our results, we ran each

fixed-time and adaptive simulation 20 times and averaged their results. Our results and interpretation for traffic throughput are illustrated below.

Quantitatively, the Webster model seems much better. The fixed-time model, on average, has a throughput of approximately 55% higher than the adaptive model; the fixed-time model had a throughput of about 0.75, while the adaptive model had a much lower throughput of 0.48. This disparity can be clearly seen in the graphs in Figure 3.

There are many interesting trends and behaviors in the data. Firstly, most of the data appears to decrease as the number of cars increases, which makes sense as there is more volume backed up and only a limited rate of exit. Overall, volumes of 100 to 300 vehicles per hour are fully cleared, meaning a 100% throughput. After the rates on both phases increase beyond 400 vehicles/hour, we start seeing significant backup.

Another important observation is that the Webster fixed-time model appears to be more efficient than the adaptive model most of the time. When we looked at the adaptive method's interval lengths (phase lengths), they tended to be shorter than those of the Webster model. This means that fewer cars will be able to clear the intersection for each traffic cycle and that the queue volume will also increase. The fixed-time model is also more "robust" than the adaptive model; that is, it is less reactive to changes in volume conditions as the curve doesn't change too drastically over time. This makes it a good traffic signal model candidate for situations where traffic volumes can vary significantly in short periods of time.

The adaptive method, on the other hand, initially appears to do much worse across the board and seems more reactive to change than does the fixed-time method. However, the adaptive method, interestingly, does better in two cases: either heavily skewed-traffic conditions (ex. 100 and 900) or, in general, when one or both are high volumes. We posit that these trends occur for the following reasons:

1. The nature of the adaptive model is forecast-based. Therefore, if there are high volume rates, the model tends to overwhelmingly favor the busy phase to the point that the less-busy phase nearly never sees a green light, or sees one only for a few seconds. In future iterations of creating a forecast-based model, this imbalance should be accounted for, potentially by implementing maximum cycle time limits.

2. Since the model overwhelmingly favors high-volume phases, low-volume phases tend to seldom turn green at all, since it is rare that the less busy phase will have a queue longer than the forecasted queue on a busier phase. This is why we see near-zero throughput for a low-volume phase in a highly skewed condition. Additionally, when less-busy phases get green lights, their intervals remain very short, for the same reason. Therefore, fewer cars clear the intersection, decreasing the throughput.

Equally important to note is the sensitivity of our results. Originally, we ran our simulations with an intersection cross time of 3 seconds. While the fixed-time method did consistently better

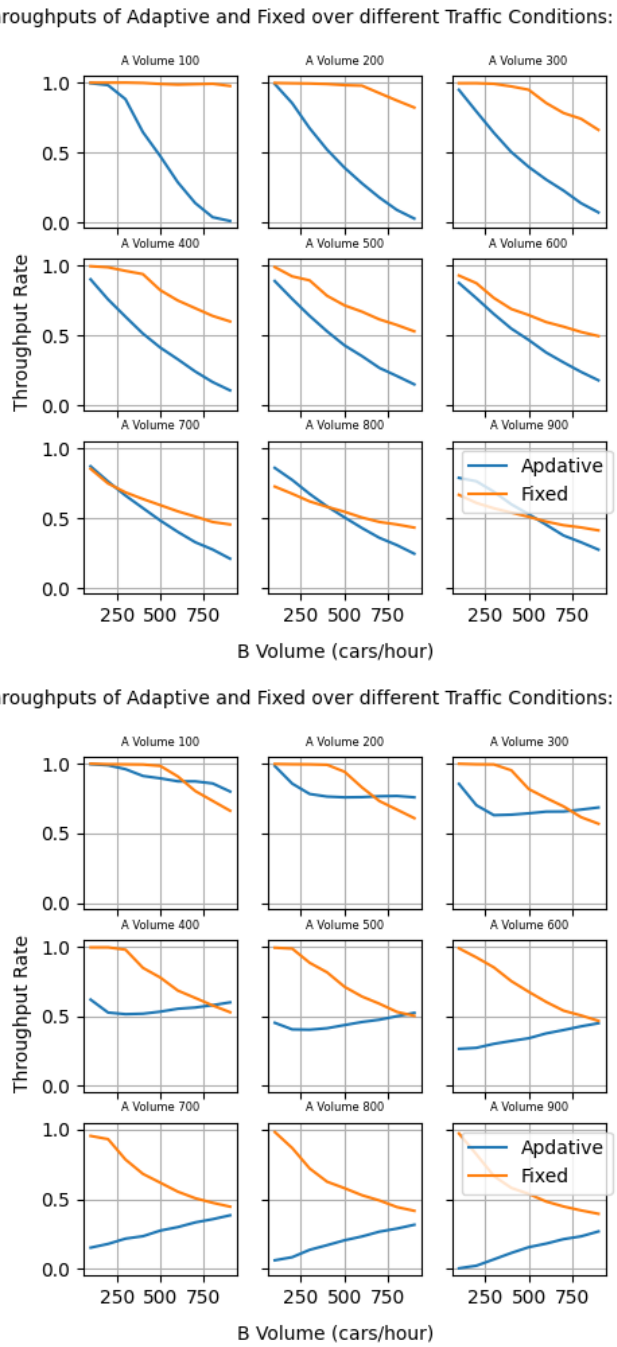

Figure 3: Phase A and B Throughput Results for Comparative Simulation

than the adaptive method and the overall trend of throughput decreasing as traffic increased existed, many of the nuanced behaviors seen by the adaptive model were not seen. Therefore, we believe that more research must be done to get accurate starting condition data (ie. intersection cross time, saturation rates, traffic rates, etc.) for the results to have meaningful insight.

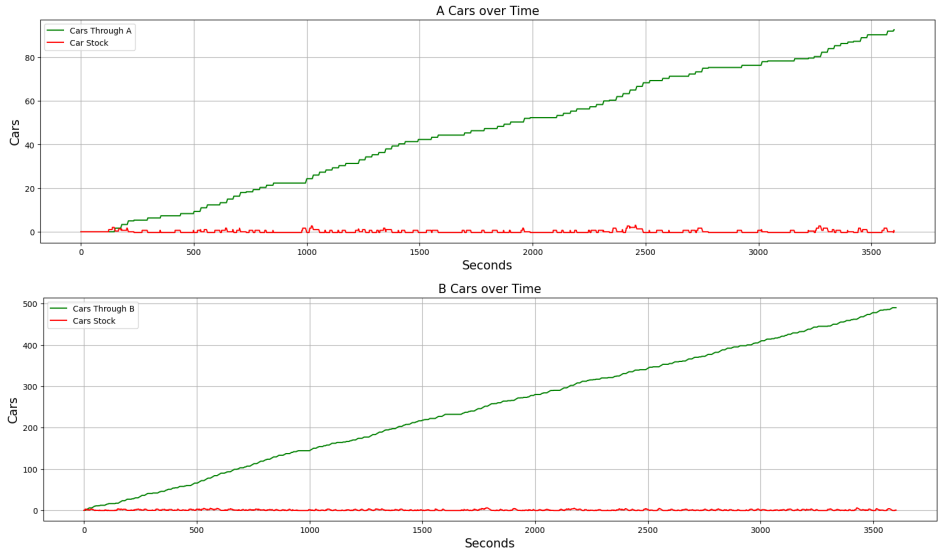

Figure 4: The throughput of Volume A = 100 cars/hr, and Volume B = 500 cars/hr

Conclusions

In this study, we modeled two traffic signal control models: one, a fixed-time model, and the other, a custom forecast-based adaptive model. By running Python traffic simulations of the two signal models and analyzing their throughput for 81 different cases, we determined that the fixed-time model tended to be much more efficient over a variety of conditions as its average throughput was 55% higher than the adaptive model's, making it the better choice over the simple forecast-based model we developed.

It is important to note, however, that the simple adaptive model performed relatively better in highly skewed traffic situations and high-volume traffic. In future iterations of forecast-based models, it will be important to maintain this quality while also increasing the throughput of lower-volume phases. This could be done by instituting a minimum phase time or limiting the green time of a higher-volume phase. Fundamentally, the issue with this forecasting model is that it prioritizes the overall throughput (ie. maximizing cars going through the intersection regardless of the phase of origin). While this may seem equitable on paper, in reality, this is an inequitable solution, as commuters on a lower-volume phase would suffer from significant delays.

In the future, the author of this paper would like

to refine their models to more accurately reflect the behaviors of actual drivers. This includes, but is not limited to, implementing variable intersection crossing times (which involves writing a function incorporating lost time and driver acceleration), all-red times, and collecting realistic data for saturation flows, traffic volumes, and intersection crossing times. Data collection will be crucial to yield realistic results, as traffic flows are highly sensitive to these parameters. Data collection will also allow us to confirm many of the results of this simulation, and help to determine the extent to which the assumptions highlighted above affect the accuracy.

Regardless, this area of study remains a critical field that holds the potential to decrease traffic idle time and increase efficiency, which could not only help commuters and prevent traffic jams and delays but also significantly decrease carbon emissions, a major contributor to climate change, and decrease resource consumption.

Acknowledgements

The author thanks Mr. Robert Gotwals for assistance with this work. Appreciation is also extended to Mr. Eric Ma (Chapel Hill High School) for providing access to the Adeniji et. al [1] paper. Finally, appreciation is extended to the North Carolina School of Science and Math Online Program, without which the development of this paper would not be possible.

References

[1] S. T. Adeniji, H. O. Ohize and U. S. Dauda, "Development of a Mathematical Model for Traffic Light Control," 2021 1st International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS), Abuja, Nigeria, 2021, pp. 1-6, doi: 10.1109/ICMEAS52683.2021.9692363.

[2] APSEd. "Traffic Signal Design - Webster's Formula for Optimum Cycle Length." APSEd, 17 July 2022, www.apsed.in/post/traffic-signal-design-webster-s-formula-for-optimum-cycle-length.

[3] "SIGNAL TIMING on a SHOESTRING - III. Signal Timing Tool Box." Ops.fhwa.dot.gov, ops.fhwa.dot.gov/publications/signal_timing/03.htm.

[4] "Capacity and Saturation Flow Rate." www.webpages.uidaho.edu, www.webpages.uidaho.edu/niatt_labmanual/Chapters/signaltimingdesign/theoryandconcepts/CapacityAndSaturationFlowRate.htm.

[5] Bester, C., and W. Meyers. SATURATION FLOW RATES. 2007, repository.up.ac.za/bitstream/handle/2263/5838/002.pdf%3Bjsessionid%3D4FCD4E6450E063E61E9B02

[6] Thete, Jueeli. "Understanding Monte Carlo Simulation and Its Implementation with Python." Medium, 12 Feb. 2022, medium.com/@juee_thete/understanding-monte-carlo-simulation-and-its-implementation-with-python-3ecacb958cd4.

[7] Liu, Jennifer. "Commuters in This City Spend 119 Hours a Year Stuck in Traffic." CNBC, CNBC, 4 Sept. 2019, www.cnbc.com/2019/09/04/commuters-in-this-city-spend-119-hours-a-year-stuck-in-traffic.html.

[8] Yau, K.-L. A., Qadir, J., Khoo, H. L., Ling, M. H., & Komisarczuk, P. (2017). A survey on reinforcement learning models and algorithms for Traffic Signal Control. ACM Computing Surveys, 50(3), 1–38. https://doi.org/10.1145/3068287

Code Appendix

The code presented here is incomplete and only shows the most important simulation methods. To access all code written for this project, including data analysis and scientific visualization, please refer to this Google Colaboratory document.

Imports

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
```

Webster's Method Function: This decides the cycle components

```
def webster_method(volume_a,volume_b,theoretical_volume_a,theoretical_volume_b,
num_phases, all_red_time):
  critical_ratio_a = volume_a/theoretical_volume_a
  critical_ratio_b = volume_b/theoretical_volume_b
  total_critical_ratio = critical_ratio_a + critical_ratio_b
  lost_time = 2*num_phases+all_red_time
  optimal_cycle_time = (1.5*lost_time+5)/(1-total_critical_ratio)
  green_a = critical_ratio_a/total_critical_ratio * (optimal_cycle_time-lost_time)
  green_b = critical_ratio_b/total_critical_ratio * (optimal_cycle_time-lost_time)
  return [optimal_cycle_time, green_a, green_b]
```

The Fixed-Time Traffic Simulation Function:

```
def fixed_time_traffic_sim(volume_a,volume_b,theoretical_volume_a,theoretical_volume_b,
num_phases,all_red_time, time_seconds, step_size,intersection_cross_rate):
  optimal_cycle_time, green_a, green_b = webster_method(volume_a,volume_b,theoretical_volume_a,
  theoretical_volume_b,num_phases,all_red_time)
  # According to the Webster method, find the optimal cycle times
  a_car_stock = [0] # Set up the queue for Phase A
  b_car_stock = [0] # Set up the queue for Phase B
  a_through = [0] # Set the cars that go through the intersection from Phase A
  b_through = [0] # Set the cars that go through the intersection from Phase B
  a_tot=0
  b_tot=0
  for i in range(time_seconds//step_size):
  # A is green at first. It's also assumed that each car takes 5 seconds to cross the
  intersection, however more advanced iterations for this simulations may see this number
  randomized.
    a=poisson_generator(volume_a,step_size)
    b=poisson_generator(volume_b,step_size)
    green = green_decider(i,optimal_cycle_time,green_a,green_b,all_red_time)
    # Returns true if Phase A is green
    a_car_stock.append(a_car_stock[i]+a)
    b_car_stock.append(b_car_stock[i]+b)
    b_through.append(b_through[i])
    a_through.append(a_through[i])
    a_tot+=a
```

```
  b_tot+=b
  if(green=="A"):
    #print("Green for A")
    a_car_stock[-1]-= intersection_cross_rate*step_size if a_car_stock[-2] >0  else 0
    a_through[-1] += intersection_cross_rate*step_size if a_car_stock[-2]>0 else 0
    #print("A"+ str(a_car_stock[-1]+a_through[-1]) +" , "+ str(a_tot))
  elif(green=="B"):
    #print("Green for B")
    b_car_stock[-1] -= intersection_cross_rate*step_size if b_car_stock[-2] > 0 else 0
    b_through[-1] += intersection_cross_rate*step_size if b_car_stock[-2]>0 else 0
    #print("B"+ str(b_car_stock[-1]+b_through[-1]) +" , "+ str(b_tot))

return optimal_cycle_time, green_a, green_b, a_car_stock, a_through, b_car_stock, b_through,
a_tot, b_tot
```

The Adaptive Traffic Simulation Function:

```
def adaptive_time_traffic_sim(volume_a,volume_b,num_phases,all_red_time, time_seconds,
step_size, intersection_cross_rate):
a_car_stock = [0] # Set up the queue for Phase A
b_car_stock = [0] # Set up the queue for Phase B
a_through = [0] # Set the cars that go through the intersection from Phase A
b_through = [0] # Set the cars that go through the intersection from Phase B
a_tot=0
b_tot=0
a_is_green = True if max(volume_a,volume_b) == volume_a else False
switch_times = []
switch_times.append(a_is_green)
lost_time = 2
for i in range(time_seconds//step_size):
  #print(i)
  a=poisson_generator(volume_a,step_size)
  b=poisson_generator(volume_b,step_size)
  a_car_stock.append(a_car_stock[i]+a)
  b_car_stock.append(b_car_stock[i]+b)
  b_through.append(b_through[i])
  a_through.append(a_through[i])
  a_tot+=a
  b_tot+=b
  if lost_time != 0: # This is the lost time. In this simulation, no all_red_time is
  assumed, though it can be easily built in later.
    lost_time-=1
  else:
    if a_is_green and traffic_forecaster(a_car_stock[i+1],b_car_stock[i+1],volume_a,volume_b,
    int(i*step_size)
    ,a_is_green, intersection_cross_rate) > int(i*step_size): # This means that A stays green,
      a_car_stock[-1]-= intersection_cross_rate*step_size if a_car_stock[-2] >0  else 0
      a_through[-1] += intersection_cross_rate*step_size if a_car_stock[-2]>0 else 0
      #print(a_through[i+1],b_through[i+1])
    elif not a_is_green and traffic_forecaster(a_car_stock[i+1],b_car_stock[i+1],volume_a,
    volume_b,int(i*step_size),a
```

```
      _is_green, intersection_cross_rate) > int(i*step_size):
      # This means that B stays green, as the forecasted switch time has not been
      reached yet.
        # print("B is green, switch time = " + str(traffic_forecaster(a_car_stock[i],
        b_car_stock[i],volume_a,volume_b,int(i*step_size),a_is_green,
        intersection_cross_time)))
        b_car_stock[-1] -= intersection_cross_rate*step_size if b_car_stock[-2] > 0 else 0
        b_through[-1] += intersection_cross_rate*step_size if b_car_stock[-2]>0 else 0
        #print(a_through[i+1],b_through[i+1])
      else:
        #print("Switch from " + "A" if a_is_green else "B")
        a_is_green = not a_is_green # Reverse the light color.
        lost_time = 2

    switch_times.append(a_is_green)


  return a_car_stock, a_through, b_car_stock, b_through, a_tot, b_tot, switch_times
```

Traffic Forecasting Function: Returns the switch time

```
  def traffic_forecaster(current_vol_a, current_vol_b, rate_a, rate_b, time, a_is_green,
  intersection_cross_time): # returns the switch time
# This method is meant to maximize cars crossing the intersection, whether it's A or B cars.
It essentially takes a starting place, and with some forecasting, figures out when the
addition of cars to the Red end
# is greater than the amount of cars released on the Green side. At this equilibrium point,
it changes lights.
  initial_time = time
  queue_time_a = current_vol_a * intersection_cross_time
  queue_time_b = current_vol_b * intersection_cross_time
  if a_is_green:
    while(queue_time_a >= queue_time_b and time < 3600):
      time+=1 # Time step for this "low accuracy method" is 1 second
      current_vol_a+=max(rate_a/3600 - intersection_cross_time,0)
      current_vol_b+=max(rate_b/3600,0)
      queue_time_a = current_vol_a * 1/intersection_cross_time
      queue_time_b = current_vol_b * 1/intersection_cross_time
    return time if initial_time != 0 else 1
  else:
    while(queue_time_b >= queue_time_a and time < 3600):
      time+=1
      current_vol_b+=max(rate_b/3600 - intersection_cross_time,0)
      current_vol_a+=max(rate_a/3600,0)
      queue_time_a = current_vol_a * 1/intersection_cross_time
      queue_time_b = current_vol_b * 1/intersection_cross_time
    return time if initial_time != 0 else 1
  return time
```