International Journal of Scientific Research in Science and Technology
Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)
doi : https://doi.org/10.32628/IJSRSET229666

Comparing Efficiencies of Different Logical Approaches of Crossing a Stop Light

Computational Physics

Rishith Singhagra
[1]Jayshree Periwal International School, Jaipur, India
Research Advisor- Dr. Andrew Haas
[2]Associate Professor of Physics at NYU College of Arts &amp; Science, New York

Article Info
Volume 9, Issue 6
Page Number : 657-664

Publication Issue
November-December-2022

Article History
Accepted : 05 Nov 2022
Published : 30 Dec 2022

ABSTRACT

As humanity has advanced, cars have become increasingly common on roads, and vehicle activity has skyrocketed. This shift has naturally created requirements for stop light systems to be implemented to control traffic flow through intersecting routes. This paper tackles the issues such as efficiency and pollution that arise due to the existence of these stop lights by exploring more efficient methods for a car to approach a stop light to save time and fuel. The different approaches are all basic logical methods tested in iterative simulations. I then found the best approach to travel from point A to point B when there is a stop light between the two points whose timing is dictated by a probability distribution.

Keywords : Probability Distribution, Stop Light

I. INTRODUCTION

Although essential to road safety, stop lights are often the source of frustration for many drivers, but more importantly, they pose a significant problem of efficiency. We are engaged in an endless search for perfectly timing traffic lights to ensure that all vehicles on-road can evenly propagate through a city's labyrinth of roads while minimizing their time idling at a stop. Idling, the act of "running a vehicle's engine when the vehicle is not in motion" (Wikipedia, 2022) at stoplights, was responsible for a total consumption of 23 billion liters of fuel and carbon dioxide emissions to the order of 30 million tons (U.S. Department of Energy, 2015).

Idle reduction is a concept that has risen to the attention of engine manufacturers and environmentalists alike; it comprises techniques and equipment that allow a combustion engine to reduce the time it runs without the vehicle being in motion. A vehicle may idle for many reasons, such as maintaining the temperature inside the vehicle, powering necessary equipment such as lighting, or waiting for a stop light to turn green. American drivers average 20% of total driving time spent idling at stop lights (Crow, 2018), making idling's impact on fuel consumption and the environment significant enough that manufacturers actively try to reduce idle time by implementing start-stop technologies, energy recovery systems, and auxiliary power systems (U.S. Department of Energy, 2022).

This paper will adapt the situation drivers face in the real world into a logical problem that, although may not directly address idling, can, to a limited extent, help simulate the conditions a car may face when approaching a stop light whose time of turning green is uncertain. Given a probability distribution for when the stop light will turn green, the model will incorporate the impacts of real-world physics, such as air resistance and varying fuel efficiencies at different speeds, and I will compare different logical approaches inspired by mathematical quantities such as mean, median, and mode, for their fuel and time efficiencies to find the most effective approach.

II. Related Work

The problem discussed has not been tackled before by any other researcher in a similar manner; however, a variety of works focus on concepts essential to the traffic system. One such paper is (Ghazal, Khatib, Chahine, & Kherfan, 2016), which addresses the problem of smoothing out the motion of cars as they pass through multiple junctions, each with its own stop light and discusses the algorithmic issue faced when having to coordinate adjacent stop lights to facilitate continuous motion for cars as well as considering occurrences such as pedestrians and emergency vehicles. They propose incorporating infrared sensors into the traffic control systems that measure the density of traffic to dynamically calculate timings for stop lights that would allow for the optimal movement of cars and other vehicles.

(Navarro-Espinoza, 2022), more recently, incorporated technologies such as MLP-NN, RNN, and various ensemble machine learning models to test their efficacy in controlling the timings of traffic lights. Their proposed methods work by dynamically predicting traffic flow based on the training from the datasets already seen to coordinate timings for traffic lights at intersections.

Most papers tackle the problem of traffic efficiency by investigating strategies to coordinate traffic lights' timings to optimize traffic flow; however, I will approach this problem by finding ways for cars to efficiently cross traffic lights while minimizing the crossing time and fuel consumption.

III.Background

3.1 Probability Distribution

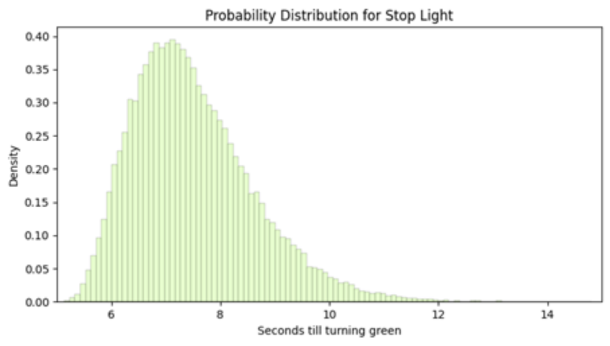
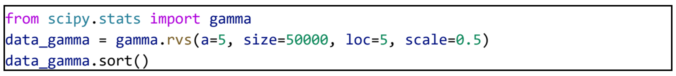


Figure 1: Probability distribution for the stoplight

Figure 1 shows the probability distribution for the stop light turning green after a certain number of seconds as a density plot. This probability distribution was created using the *gamma* function from the *scipy* module for Python 3.10 using the following code.



```
from scipy.stats import gamma
data_gamma = gamma.rvs(a=5, size=50000, loc=5, scale=0.5)
data_gamma.sort()
```

Figure 2: Code used to generate gamma distribution

The gamma function was chosen as it can create a distribution that is not purely random at all values but provides a mounded probability distribution to which logical approaches can be applied.

Table 1: Hyperparameters used in the gamma distribution

Hyperparameter	Purpose	Value
a	Shape parameter that shifts the size and location of 'mound'.	5
size	Number of samples to be taken from gamma distribution	50 000
loc	Starting position of gamma distribution, in other words, how much it is shifted from the origin.	5
scale	Scale factor to be applied to the original gamma distribution.	0.5

Table 1 showcases the hyperparameters used while creating the gamma distribution and their impact on the final distribution generated.

The final probability distribution was a sorted list of 50 000 samples from a gamma distribution with times that ranged from 5 seconds to 15 seconds. The distribution was taken as a set of samples rather than a continuous equation to facilitate finding mathematical quantities such as mean, median, and mode while running the different algorithms.

3.2 Mathematical Quantities

The following mathematical quantities inspire the algorithms used:

- **Mean**: The sum of all values in a list divided by the number of elements in the list.
- **Median**: The value present at the halfway point when linearly traversing an ordered list of numbers of ascending value.
- **Mode**: The value most common / appearing the most in a list.

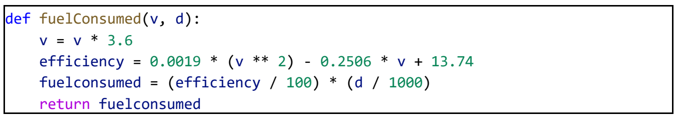
Table 2 : Mathematical quantities calculated

Name of Quantity	How to Calculate
average	Finds the mean of a given list
averagep10	Finds the mean of a given list and adds 20% of the difference between the maximum and mean value to it
averagem10	Finds the mean of a given list and reduces 20% of the difference between the minimum and mean value from it
median	Finds the median value in an ordered list
medianp10	Finds the value found at the position 60% into the list
medianm10	Finds the value found at the position 40% into the list
peak	Divides the list into a histogram with 100 bars of equal interval and returns the average of the interval of the bar with the highest density
peak2	Divides the list into a histogram with 100 bars of equal interval and returns the average of the interval of the bar with the second highest density
peak3	Divides the list into a histogram with 100 bars of equal interval and returns the average of the interval of the bar with the third highest density
minimum	Returns the maximum value of the list

Table 3 shows how different quantities will be calculated for the algorithms that will be run. Each quantity will be calculated based on an inputted list of numbers.

3.3 Additional Functions

Fuel Efficiency



```
def fuelConsumed(v, d):
    v = v * 3.6
    efficiency = 0.0019 * (v ** 2) - 0.2506 * v + 13.74
    fuelconsumed = (efficiency / 100) * (d / 1000)
    return fuelconsumed
```

Figure 3: Code used to calculate fuel consumption

Figure 3 shows the function *fuelConsumed* that returns the amount of fuel consumed in liters, *fuelconsumed*, when given the car's speed in meters per second, *v*, and the distance traveled in meters, *d*, as inputs. This function is a python implementation of the following fuel efficiency equation (Tarulescu & Tarulescu, 2016):

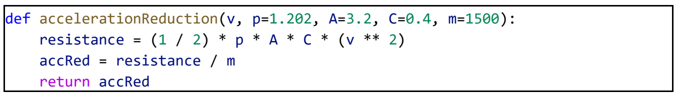
$$y = 0.0019x^2 - 0.2506x + 13.74$$

Equation 1: Equation for fuel efficiency of a car

In Equation 1, y represents the fuel efficiency of a car in liters consumed per 100 kilometers driven, where x is the vehicle's average speed in kilometers per hour.

In the function, the fuel efficiency is converted to liters per kilometer and multiplied by the distance traveled in kilometers to find the total fuel consumed.

Air Resistance



```
def accelerationReduction(v, p=1.202, A=3.2, C=0.4, m=1500):
    resistance = (1 / 2) * p * A * C * (v ** 2)
    accRed = resistance / m
    return accRed
```

Figure 4: Code used to calculate air resistance

Figure 4 shows the function *accelerationReduction* that returns the reduction in the maximum acceleration of a car traveling at speed v in meters per second due to air resistance. This function is a python implementation of the following air resistance equation (Car Performance Formulas, 2022):

$$AR = \frac{1}{2}\rho A_f C_d \left(\frac{v}{3.6}\right)^2$$

Equation 2: Equation for air resistance force on a car

In Equation 2, AR represents the air resistance force, in Newtons, on a car where ρ is the air density in kilograms per meter cubed, A_f is the total area of the car that is perpendicular to relative wind velocity and is in a frontal position, C_d is the drag coefficient, and v is the car's speed in kilometers per hour. This equation is valid if wind velocity is negligible. For the car in this experiment, the air density was set to 1.202 kilograms per meter cubed, the frontal car area was set as 3.2 meters squared, the drag coefficient was set to 0.4, and the mass was decided as 1500 kilograms.

The resistive force that is found using the equation above is then used to calculate the reduction in maximum acceleration using Newton's second law:

$$a = \frac{F}{m}$$

$$\therefore a = \frac{F}{1500}$$

Equation 3: Equation for the reduction in acceleration using air resistance

The value found for the reduction in acceleration is then stored in a variable, *accRed*, and returned to the function caller.

Technical Approach

The issue to approach is as follows: there is a car that starts at Point A and must reach point B in the most time-efficient and fuel-efficient manner; however, there is a stop light at a fixed position in between the two points whose time of turning green is unknown and dictated by a set probability distribution. The car can dynamically accelerate and decelerate, but it can not have a velocity of less than 0 m/s at any point, meaning it can only move in the forward direction. The first condition during runtime is that the algorithm must end, and the timer must stop when the car reaches or passes point B. The second condition is that when the light turns green, the car must accelerate until it reaches its maximum velocity and continues traveling at maximum velocity until it reaches its destination.

For this simulation, Point A is located at 0 meters, Point B is located at 80 meters, and the traffic light is placed halfway between both points at 40 meters. The car can accelerate at a maximum rate of 2.87 meters per second squared at 0 Newtons of air resistance, decelerate at a maximum rate of 4.57 meters per second squared, and achieve a maximum speed of 22.22 meters per second, which is equivalent to 80 kilometers per hour.

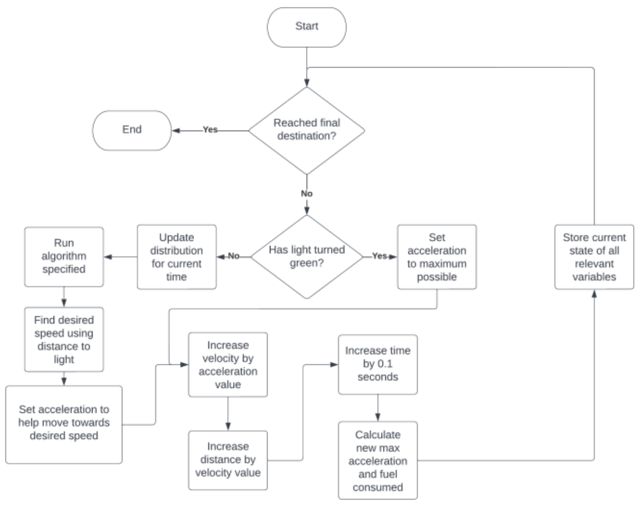

Start
Reached final destination?
Yes
End
No
Has light turned green?
No
Update distribution for current time
Run algorithm specified
Find desired speed using distance to light
Set acceleration to help move towards desired speed
Yes
Set acceleration to maximum possible
Increase velocity by acceleration value
Increase distance by velocity value
Increase time by 0.1 seconds
Calculate new max acceleration and fuel consumed
Store current state of all relevant variables

Figure 5: Flowchart for running one simulation

Figure 5 is a flowchart that shows how a simulation is run. It can be observed that the simulation moves in time steps of 0.1 seconds, meaning that new acceleration, velocity, and displacement are calculated, and the current simulation time is increased by 0.1 seconds for each iteration of the loop.

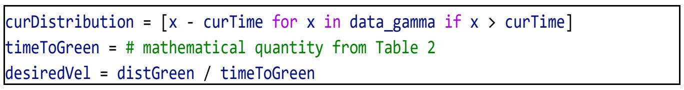

```
curDistribution = [x - curTime for x in data_gamma if x > curTime]
timeToGreen = # mathematical quantity from Table 2
desiredVel = distGreen / timeToGreen
```

Figure 6: Code used to update distribution each time it loops

While the light is still red, the simulation works by taking the original list of 50 000 values sampled from the gamma distribution and removing all values that happen before the current simulation time, then subtracting the simulation time to get a final current distribution, shown by the variable named *curDistribution*, which is a list of values that represent a probability distribution of the light turning green after that many seconds from the current simulation time. *curDistribution* is then passed as an input to one of the various functions defined to find a mathematical quantity listed in *Table 2,* depending on which one was chosen for the simulation. Using the distance left till the stop light, *distGreen*, and the value assigned to *timeToGreen*, the car's velocity should aim to reach that moment is calculated.

Once *desiredVel* is calculated, the algorithm executes a series of logical comparisons to find the acceleration that will help reach the desired velocity the quickest after considering maximum acceleration, maximum deceleration, maximum speed, and minimum speed. The velocity is then updated based on the new acceleration, and the displacement is updated based on the new velocity. The fuel consumption in that 0.1 seconds of simulation is calculated and added to the running total, the time is then incremented by 0.1 seconds, and all data about the car is stored.

Air resistance is considered by calculating the acceleration reduction for every iteration of the loop using the *accelerationReduction* function with the last recorded velocity as input and overwriting the max acceleration value by subtracting what the function returned from 2.87 meters per second, the max acceleration when the car is at rest.

This process is repeated until the light turns green, and a standard loop that makes the car accelerate to full speed until it reaches the destination is executed.

IV. CONCLUSION

In our study, the results showed that PRP is a safe and effective treatmentmodality for chronic non-healing ulcers. Decrease in pain was observed in post PRP treatment.Delivering of growth factors to target site enhances the wound healing rates of chronic non healing ulcers.PRP seems to be efficient to treat chronic non healing ulcers which are non responsive to classical conservative treatments. Using PRP to treat chronic wounds/ulcers may not only enhance healing, but also prevent lower extremity amputations caused by nonhealing wounds.There by reducing over all hospital stay, inconvenience of constant medication and morbidity.

V. Experiments

5.1 Experimental Setup

I ran the algorithms below using Python 3.10 on Windows 11 version 22H2 on a ROG Strix G533QS_G533QS, which houses 32GB of random-access memory and an 8-core AMD Ryzen 9 5900HX clocked at 3.3 GHz. The module *matplotlib* was used to visualize the data collected, *pickle* was used to store the data collected, *numpy* was used to create data structures and perform calculations, *scipy* was used to generate a probability distribution, *random* was used to choose times for the stop light randomly, and *pyinstaller* was used to compile the python code into an executable.

5.2 Average Results

Table 3: Average results for all ten algorithms

Algorithm Used	Average Time to Reach Destination (seconds)	Average Fuel Consumed (ml)
average	12.15	6.686
averagep10	12.29	6.674
averagem10	12.24	6.673
median	12.16	6.678
medianp10	12.11	6.67
medianm10	12.23	6.668
peak	11.58	6.492
peak2	11.86	6.553
peak3	12.04	6.575
minimum	13.17	6.45

Table 3 displays the average results obtained for the ten different approaches over 1756 simulations for each algorithm. The results show that the approach with the fastest average time was *peak*, followed by *peak2* and *peak3*. The results also show that the most fuel-efficient algorithm was *minimum*, followed by *peak* and *peak2*.

Table 4: Composite scores for algorithms

Algorithm	Time-Efficiency Score	Fuel-Efficiency Score	Composite Score
average	0.642	0	0.642
averagep10	0.553	0.051	0.604
averagem10	0.585	0.055	0.64
median	0.635	0.034	0.669
medianp10	0.667	0.068	0.735
medianm10	0.591	0.076	0.667
peak	1	0.822	1.822
peak2	0.824	0.564	1.388
peak3	0.711	0.47	1.181
minimum	0	1	1

Table 4 shows the Time-Efficiency Score (TES) and Fuel-Efficiency Score (FES) for each algorithm, which was calculated by scaling all scores in a way where the highest reading was assigned a value of 0 and the lowest reading was assigned a value of 1. A composite score was derived by adding both the TES and FES values to show that the best algorithm for approaching this problem is *peak* with a composite score of 1.822, followed by *peak2* with a score of 1.388, and *peak3* with a score of 1.181.

5.3 Edge Cases

Although average results are a fair indicator of which algorithm will perform better, analyzing each algorithm's behavior in edge cases is also essential to identify an approach's strengths and weaknesses.

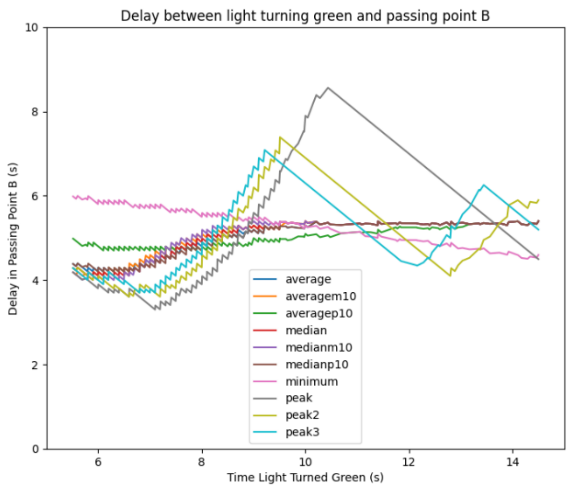


Figure 7: Graphing performance of all algorithms at different times

Table 5: Maximum and minimum results

Algorithm	Minimum Crossing Delay (s)	Maximum Crossing Delay (s)	Crossing Delay Range (s)
average	4.2	5.4	1.2
averagep10	4.7	5.4	0.7
averagem10	4.1	5.4	1.3
median	4.1	5.4	1.3
medianp10	4.2	5.4	1.2
medianm10	4.0	5.4	1.4
peak	3.3	8.6	5.3
peak2	3.6	7.4	3.8
peak3	3.7	7.1	3.4
minimum	4.5	6.0	1.5

Figure 7 and Table 5 measure the time between the stop light turning green and the car passing Point B. As visible in Figure 7 and Table 5, *peak* may have been the algorithm with the highest average composite scores but is also the most unreliable, with the difference between the best-case scenario and worst-case scenario for the algorithm being 5.3 seconds. Similarly, the algorithm with the worst composite score is the most reliable of the 10, with a range in crossing delay of just 0.7 seconds. This result shows that because of the 'mound' in the probability distribution, a volatile algorithm like *peak* can generate phenomenal average results but also have specific performances far worse than any other.

VI. Conclusion

Through the findings in this paper, I concluded that *peak* is likely the best algorithm to use if approaching a traffic light with a given probability distribution and wish to achieve the fastest average time; however, if you want to ensure consistency and reliability even when luck is not on your side, one of the other models with a lower crossing delay range would fit your requirements better.

One way to improve the effectiveness of the algorithms would be to extract ranges where specific algorithms work best and create a blend of 2 or more approaches used in the paper based on the distribution during the current simulation time.

One problem with the simulation can be observed in Figure 7: The plots are very jagged and have slight increases and decreases in the form of zigzags. These are caused by the simulation working in 0.1-second timesteps while the stoplight can turn green at a far more continuous level with up to 10 decimal points of precision. A solution to this would be to increase time step resolution to something like 0.01 seconds, making the graph much smoother to the eye and more accurate for analytical purposes.

This same problem could also be approached with more modern techniques such as machine learning, which would not limit us to set algorithms to follow or quantities to find; instead, the model could be used to dynamically find optimal acceleration values using highly complex approaches that cannot be logically outlined.

Although the direct real-world application of the research in this paper is limited, as rarely are these ideal conditions met, such as no congestion before the traffic light. In addition, a human driver cannot compute the different quantities required to take these approaches accurately. Therefore, research on different methods for cars to approach stop lights in more real-world settings, such as dealing with congestion, pedestrians, or even multiple traffic lights, or in a way that a human driver could execute may help in devising better systems to control the flow of traffic on roads as well as minimize the problems associated with idling and wasting of time and velocity at stop lights.

VII. REFERENCES

[1]. Crow, S. (2018, September 19). You'll Spend This Much of Your Life Waiting at Red Lights. Retrieved from yahoo!: https://www.yahoo.com/lifestyle/ll-spend-much-life-waiting-140236110.html

[2]. Ghazal, B., Khatib, K., Chahine, K., & Kherfan, M. (2016). Smart traffic light control system. Beirut, Lebanon: IEEE.

[3]. Navarro-Espinoza, A. L.-B.-G.-C.-M.-M.-G. (2022). Traffic Flow Prediction for Smart Traffic Lights Using Machine Learning Algorithms. Mexico: Technologies.

[4]. U.S. Department of Energy. (2015). Idling Reduction for Personal Vehicles. Chicago: Argonne National Laboratory. Retrieved 09 28, 22, from https://afdc.energy.gov/files/u/publication/idling_personal_vehicles.pdf

[5]. U.S. Department of Energy. (2022, September 28). Medium-Duty Vehicle Idle Reduction Strategies. Retrieved from Alternative Fuels Data Center: https://afdc.energy.gov/conserve/idle_reduction_medium.html

[6]. Wikipedia. (2022, September 28). Idle (engine). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Idle_(engine)

Cite this article as :

Rishith Singhagra, Dr. Andrew Haas, "Comparing Efficiencies of Different Logical Approaches of Crossing a Stop Light", International Journal of Scientific Research in Science and Technology (IJSRST), Online ISSN : 2395-602X, Print ISSN : 2395-6011, Volume 9 Issue 6, pp. 657-664, November-December 2022. Available at doi : https://doi.org/10.32628/IJSRSET229666
Journal URL : https://ijsrst.com/IJSRSET229666