Traffic Model

New Mexico Supercomputing Challenge

Final Report

April 3, 2019

Rishi Tikare Yang

Team Number: 4

School Name: Albuquerque High School

Teacher: Joan Newsom

Executive Summary

Traffic flow in cites affects nearly everyone's lives as our society depends on automobiles for transportation. Making traffic more efficient could have effects ranging from increased productivity of the city's inhabitants, to reducing CO_2 emissions from cars. For my project, I have modeled traffic focusing on the questions: What variables have the greatest impact on traffic flow? How can we make traffic flow more efficient?

My model, developed in NetLogo[1], consists of roads on a rectangular grid. Each road is two lanes, with one going in each direction and stop lights at intersections. My model can control the following variables: stoplight cycle duration, the timing of lights relative to each other, car follow distance, car spawn rate and car speed. My model tracks the percentage of cars waiting at a given time and the average trip duration of the cars in the model. In this report, I will describe my model, show results and discuss the effects and significance of each variable.

The only variable that could approach zero wait time was stoplights cycle duration. The stoplight timing and cycle length could be optimized by timing lights according to the speed and distance between intersection. With this timing, no cars waited after they entered the road system. Theoretically this timing can be applied to any grid-like road system. Car follow distance was only significant after a threshold which was dependant on car density.

My model can be applied to roads in our city to assist in optimizing the road system. My model also can predict how advances in technology, such as self driving cars that can drive in a precise manner, could increase or change efficiency of our roads, and how roads should be designed to most effectively work with self-driving cars.

---

[1] Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

Problem Statement

Almost everyone has to deal with traffic on a daily basis. Slow traffic and badly timed lights are frustrating and waste time from everyone's day. Not only does inefficient traffic waste time, it increases pollution released from vehicles. A typical passenger vehicle emits about 4.6 metric tons of carbon dioxide per year[2], and it is predicted there will be 281.3 million registered vehicles in the U.S. this year.[3] Optimizing roads can help reduce the amount of fuels burned, reduce wasted time of transportation, boost productivity, and save drivers money.

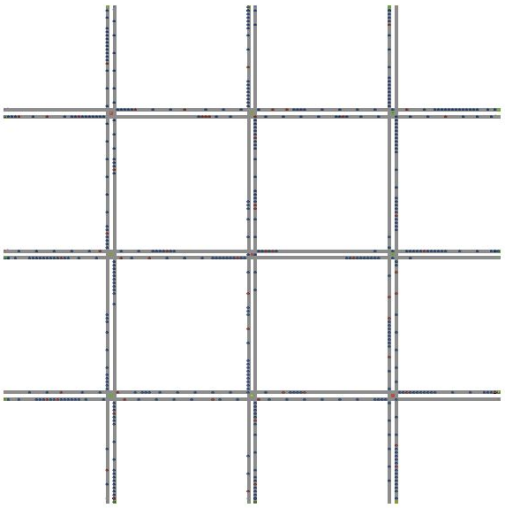
Solution Method

I have developed a model to test which variables affect traffic flow efficiency and to what extent. I have created a NetLogo model that simulates cars driving through a road system. My model is able to control variables of the cars, road, and stop lights, and tracks data on the efficiency of vehicles moving through the system. By systematically changing my variables and analyzing the results, I was able to determine how variables influence efficiency and to what extent.. I created simulations that would collect data on the model, and gradually increase or decrease a variable to chart the change in efficiency. This method proved effective and allowed me to visualize and find trends of a changing variable.

Model Description

[2] https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle
[3] https://hedgescompany.com/automotive-market-research-statistics/auto-mailing-lists-and-marketing/

My model consists of the a road system (see figure below), stoplights, and cars. Car position and stoplight cycles are updated every tick, which is the unit of time. Each lane of a roads has a set direction that cars follow. At every tick, cars move ahead a set distance which simulates *car speed*. Cars will only move if there are no cars within a given distance in front, which I call *car follow distance.* If a car is stopped at an intersection, it reports it is *waiting*, and if a car approaches from behind a waiting car, it will stop right behind it, and then is also *waiting*. The *percent of cars waiting* is collected and plotted. At every tick, cars have a probability of spawning at the beginning of each road, which I named *car spawn probability.* When a car reaches the end of a road, it is despawned and reports the number of ticks it was alive, or it's *trip duration*. The average trip duration is plotted. Stoplights are set to allow traffic horizontally or vertically for a certain number of ticks, which I call *stoplight cycle duration*. All stoplights in a simulation have the same stoplight cycle duration because all intersections are equidistant from each other. The timing of stoplights switching directions relative to each other can be controlled, and is called *relative stoplight timing*.

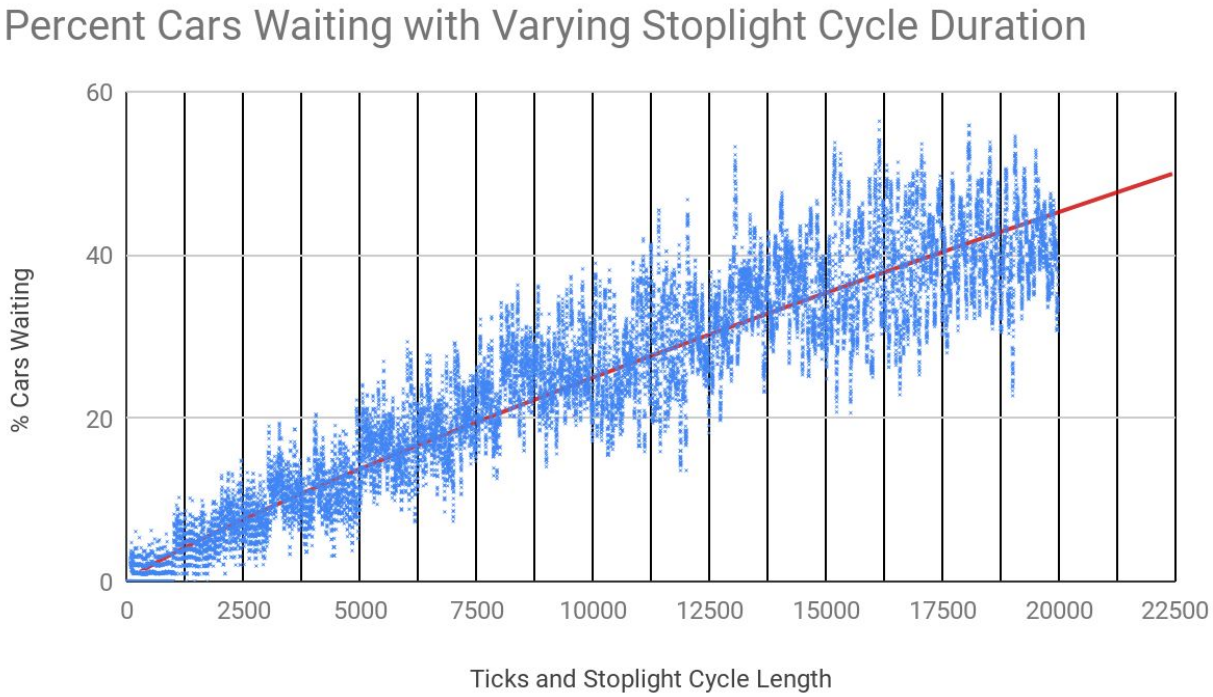


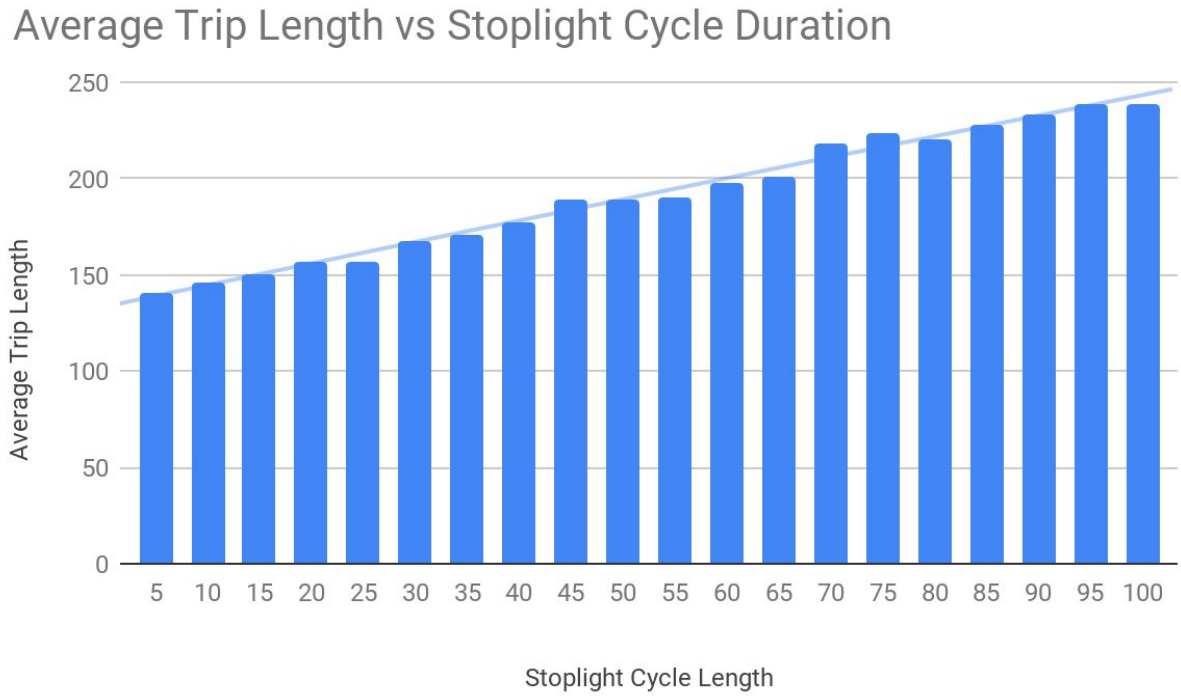
Validation

Cars and drivers follow specific patterns and rules of traffic, and by using agent-based modeling to program agents to follows these same rules, my model can prove accurate and useful in finding trends in traffic flow as a function of model variables.. An example is having cars follow a set distance behind the car ahead, which

more accurately simulates cars accelerating out of an intersection. I have also considered limitations in my model and tried to account for them while running simulations, as well as taken care to completely isolate variables while testing. An example is during my simulations, the calculated average trip duration of car in my model only averaged cars that completed their trip in the last half of each interval, so that any adjustment period cause by a changed variable does not influence my data. My model has limitations; it only has two lane roads, and does not consider human behavior and variation in driving. A specific limitation is that NetLogo moves each car one at a time, so cars have instantaneous reactions to cars ahead of them and to stoplights. This allows my model to operate without stoplights altogether. This clearly is not how people drive, but perhaps can be applied to self driving cars, where ideally every car in the system can communicate and have accurate predictions of the positions of other cars, and have instantaneous reactions.

Results

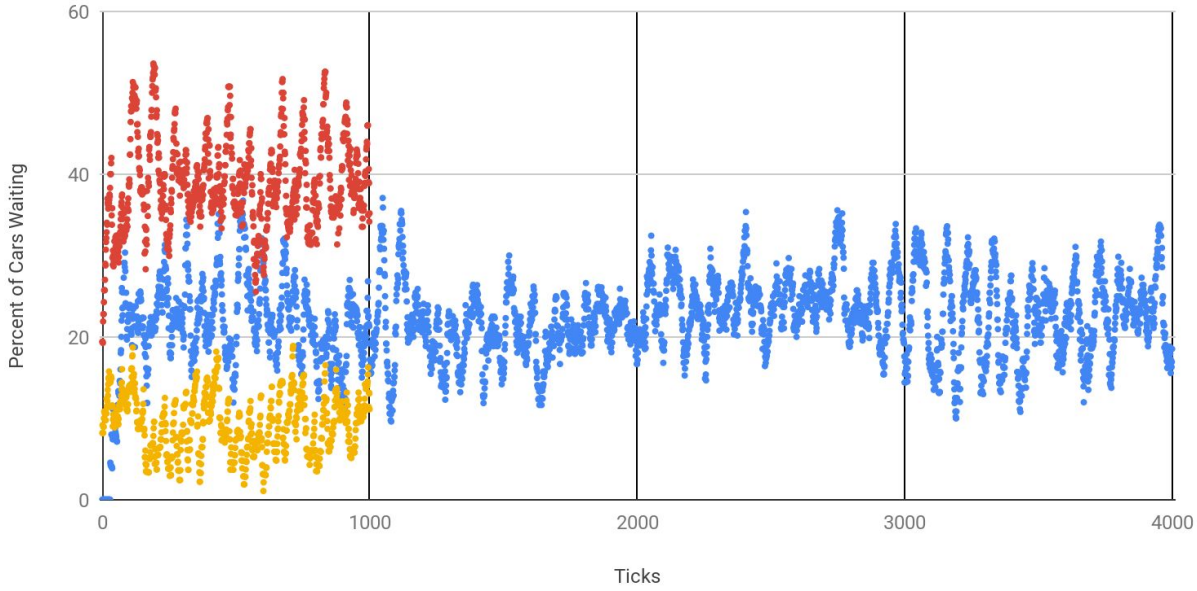

Percent Cars Waiting with Varying Stoplight Cycle Duration
% Cars Waiting
60
40
20
0
0
2500
5000
7500
10000
12500
15000
17500
20000
22500
Ticks and Stoplight Cycle Length


Average Trip Length vs Stoplight Cycle Duration
Average Trip Length
250
200
150
100
50
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
Stoplight Cycle Length

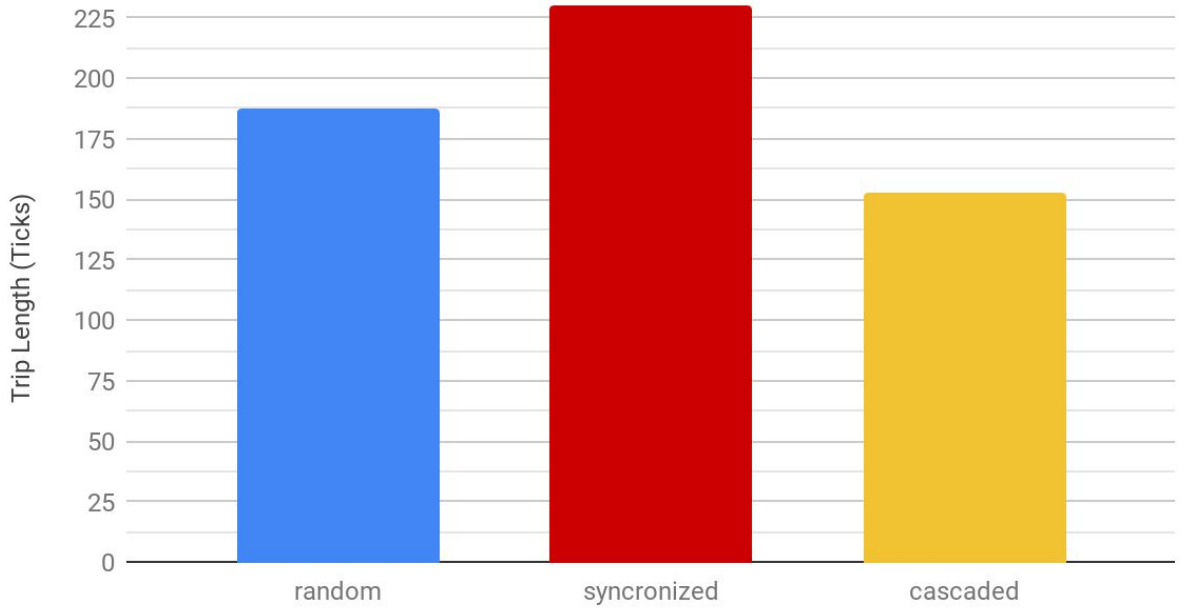
To collect data from my simulation, I used the plot feature in NetLogo that creates a graph and collects data that I specify on intervals I specify. I then exported the data into the spreadsheet software Google Sheets for analysis and visualization. For this simulation, I varied **stoplight cycle duration**. I initialized my model starting the stoplight cycle length to 5 ticks, randomized the relative light timing, and all other variable were held constant. Every 1000 ticks, the stoplight cycle length increased by 5 ticks and the relative light timing was re-randomized. The percent of cars waiting was collected every tick and the average trip duration was found from the cars in the last half of each interval. The data in the graph was an average of a couple simulation runs. The first graph plots the percentage of cars waiting at every tick as blue dots. The red line is a trend line fit to all these data points. The percentage of cars waiting and the variation in cars waiting per tick increases with stoplight cycle duration. The second graph shows the average trip duration correspondingly increased with stoplight cycle. The data points show wait times increased almost linearly with stoplight cycle length, with a slight concavity down. This simulation ignores the fact that in real traffic, with shorter stoplight cycles drivers must more accurately predict when to accelerate and stop. My model assumes all drivers are always paying attention and react instantaneously. This means my model is not an accurate predictor of human drivers at the very short stoplight cycles, but my model might be able to predict how self-driving cars would behave under these conditions. The trend that shorter stoplight cycles reduces wait times is the obvious conclusion.

Percent of Cars Waiting vs. Ticks Comparing Relative Stoplight Timing

Blue(random) Red(Synchronized) Yellow(Cascaded)



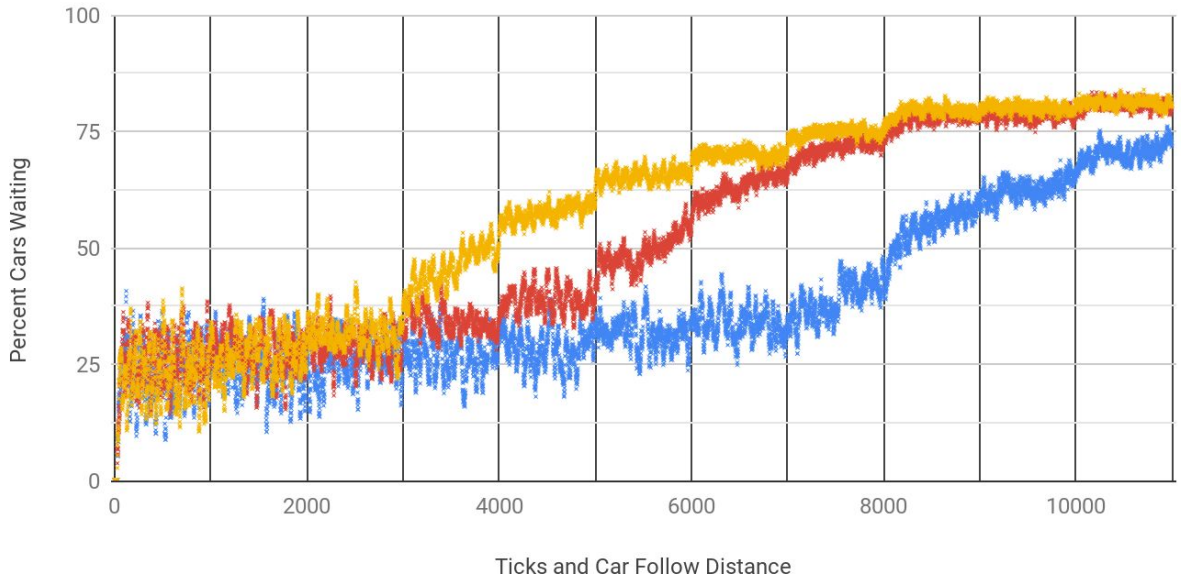
Average Trip Length vs. Relative Stoplight Timing



For the next simulation, I varied the relative **stoplight cycle timing**. The stoplight cycle duration was held constant at 40 ticks for the entire simulation, because this is the most efficient cycle length for cascaded lights. During this simulation, I ran the model for 1000 tick intervals of randomized, synchronized, or cascaded lights. I ran 4 intervals of different random stoplight timings, and found the average trip length over all 4. The randomizing code initialized the stoplights to randomly select between vertical or horizontal traffic, and randomly set the number of ticks -between 0 and the stoplight cycle duration- until it changes directions. This means all the lights cycle with the same cycle duration, but are randomly offset in direction and timing. The synchronizing code initialized the lights to all be in horizontal traffic, and all change direction on the same cycle. The code to cascade the lights initialized the stoplights to allow vertical traffic, and then were timed so that adjacent lights switched 40 ticks later, which is exactly the time it took for a car to travel from one intersection to the other. Because the cycle duration was equal to the travel time between stoplights, by cascading in to the right and down directions, the system it also is cascaded left and up. This simulation shows that with predictable travel times between stop lights, roads can be optimized to have the same stoplight cycle duration, and can be cascaded in all directions. The limitation of my model is again that it ignores human reaction time. At short stoplight cycle durations, human reaction and prediction would most likely be too imprecise and inconsistent to function at maximum efficiency, and human drivers do not have perfectly predictable travel times between stoplights. The most significant conclusion of this test it that it is possible to time lights on a grid pattern of roads to allow drivers to never stop at intersection once they hit a green light. The limitation to this it that cycle required might be too low to be efficient with human drivers, or with very dense traffic.

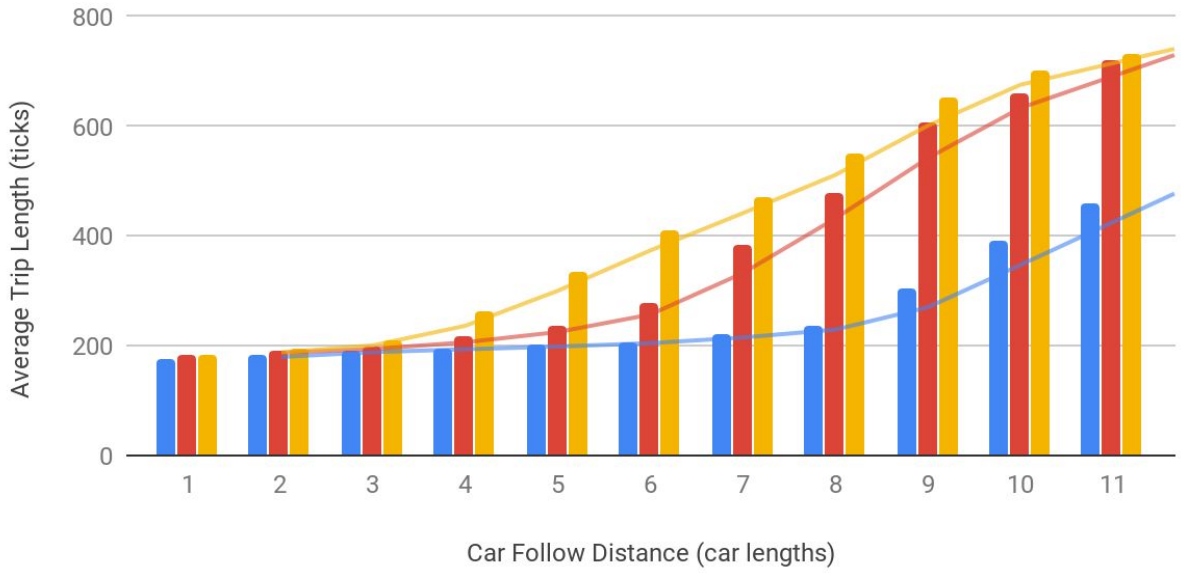
Percent of Cars Waiting with Varying Car Follow Distance

Spawn Probability: Blue (6%) Red (9%) Yellow (12%)



Average Trip Length vs. Car Follow Distance

Spawn Probability: Blue (6%) Red (9%) Yellow (12%)



In the next simulation I varied the **car follow distance**. I initialized the model to a stoplight cycle duration of 50 ticks, initially randomized relative light timing, and set the car spawn probability to 6%. Then, starting at 1, increased the car follow distance every 1000 ticks. Again, I collected the percent of cars waiting per ticks, and the average trip duration of every interval. To ensure the average was only influence by the current variables, I found the average trip duration of cars in the last half of each interval, to give the model time to cycle through the transition periods between varying car follow distances. I ran this simulation at 3 different car spawn probabilities, shown in blue, red, and yellow, with car spawn percentages of 6%, 9%, and 12% respectively. The data suggests that there is a threshold, that depends on the amount of cars on the road, where below the threshold the car follow distance is not very significant, but over the threshold wait times increase significantly with car follow distance. During the blue test (6% car spawn probability), that threshold is around 8 car length, in the red test (9% car spawn probability) the threshold is around 5 car lengths, and in the yellow test (12% car spawn probability) the threshold is around 3 car lengths. There also appears to be a upper limit to how significant car follow distance can be, but I believe this is because my model will only spawn cars as fast as they can be spawned without backing up behind the spawn points. I believe this upper threshold is caused by this limitation of my model. But in real traffic, there is also no reason to drive such a large distance behind the car ahead of someone in the conditions my model simulates.

Conclusions

My model established trends of traffic efficiency for a changing variable, including stoplight cycle duration, relative stoplight light timing, and car follow distance.

Stoplight cycle duration increased wait times almost linearly, with a slight concavity downwards. Stoplight cycle duration was the only variable that had a lower limit of 0 percent of cars waiting, as the stoplight cycle duration approached 0. The range of variation in percent of cars waiting also increased with cycle duration, showing the larger and longer cycles caused by the stoplights cycle. The shortest stoplight cycle durations are not valid for human drivers. My model moves each car one at a time after checking the spaces ahead is not occupied. This means the model operates so that cars react instantaneously and can drive in a precise manner, which humans cannot do reliably. My model may still be valid for self-driving cars, and can be used optimize stoplights for computer controlled cars that can both react instantaneously and predict other car movement with precision.

Relative stoplight timing had a significant effect on the efficiency of traffic. The three types of light timing my model implements are randomized, synchronized, and cascaded. Randomized timing was fairly consistent, and in simulations where multiple random light timings were tested, their average trip durations had little variation. The percent of cars waiting had varying cycles and ranges, but had similar averages. Synchronized lights were less efficient than randomized lights, and only were more efficient at specific stoplight cycle durations where it mimicked cascaded light timing. Cascaded lights were the most efficient timing of the three at specific stoplight cycle durations. Cascaded lights only required cars that were entering the system to wait for the first green light. When creating the algorithm to cascade lights, I learned that in a grid pattern, it is possible to cascade all roads in both directions if the stoplight cycle duration was equal to the travel time between intersections, or half the travel time.

An interesting insight I reached through creating this algorithm is that for any road system where only two roads meet at any intersection, there is a possible cascading light timing that where no car will wait at any stoplight (except the initial light). This would required the stoplight cycle duration to be equal to the lowest common factor of all travel times between all intersections, and cars to travel between intersections in precise times. This is not always possible for human drivers in city road systems because it would require, most likely, a very short stoplight cycle duration and precision in driving time.

Car follow distance had a distinct threshold;below this threshold the follow distance had little effect on efficiency, but above this threshold follow distance increasingly reduced efficiency. This threshold depended on the number of cars on a road, or the car density. The threshold at which follow distance became significant was lower with higher car density. In my simulation I achieved higher car density by increasing the car spawn rate. In the graph of average trip duration versus follow distance, there is a very shallow slope before this threshold, and after a steep slope. My model shows an upper limit where longer car follow distances would not make traffic more inefficient, and this limit was consistent regardless of car density. This conclusion is not valid because of a limitation of my model, which would stop spawning cars if traffic backed up to spawn points.

Achievements

I created my first numerical model, and used it to draw meaningful conclusions that could be applied to our world. I learned how model-based simulations allow one to gain insights into complex physical systems.

Acknowledgements

I would like to thank my teacher Ms. Newsom who introduced me and guided through the Supercomputing Challenge. I would like to acknowledge the organizers of the Supercomputing Challenge who provide a wonderful service to our community. I would like to thank my parents for helping me proofread my work and general guidance.

Appendix

Code:

```
#main code

breed[cars car]
cars-own[go-dist maxspeed acceleration tic tics-alive follow-dist waiting?]

breed[intersections intersection]
intersections-own[up? down? right? left? ]

directed-link-breed[redlinks redlink]

patches-own[direction direction? intersection?]

breed[stoplights stoplight]
stoplights-own[vert? switch maxswitch]

globals[cars-waiting total-trip-length num-of-cars-despawned selected-stoplight
num-of-sums CWS]

to setup
  clear-all
  reset-ticks
  setup-colors
  ;set CWS 0 set num-of-sums 0
  set total-trip-length 0 set num-of-cars-despawned 0
  make-roads
  setup-roads
```

```
  initialize-stoplights
end

to go
  if spawn-cars? [spawn-cars]
  move-cars
  work-stoplights
  despawn-cars
  monitor-cars-waiting
  ask cars [set tics-alive (tics-alive + 1)]
  if display-timing? [display-timing]
  tick
end


to setup-colors

    ask patches [set pcolor white]
    ask intersections [set color yellow set shape "flag"]

end
to make-roads
  ask patches [set intersection? false]

  make-intersections 70 1 -70 1
  make-intersections -70 -1 70 -1

  make-intersections 70 41 -70 41
  make-intersections -70 39 70 39

  make-intersections 70 -39 -70 -39
  make-intersections -70 -41 70 -41

  make-intersections 1 -70 1 70
  make-intersections -1 70 -1 -70

  make-intersections 41 -70 41 70
  make-intersections 39 70 39 -70

  make-intersections -39 -70 -39 70
  make-intersections -41 70 -41 -70

end

to initialize-stoplights
```

```
  make-stoplight 40 40
  make-stoplight 40 0
  make-stoplight 40 -40
  make-stoplight 0 40
  make-stoplight 0 0
  make-stoplight 0 -40
  make-stoplight -40 40
  make-stoplight -40 0
  make-stoplight -40 -40

   ask stoplights
  [
    set vert? false
    set maxswitch stoplight-cycle-length
    set switch random maxswitch
  ]
  randomize-lights
end

to make-stoplight [x y]
  ask patch x y [sprout-stoplights 1 ]
  ask patches with [pxcor > (x - 2) and pxcor < (x + 2) and pycor > (y - 2) and pycor < (y + 2)]
[set direction "fd"]


end

to make-intersections [x y xx yy]

  make-intersection x y
  make-intersection xx yy
  connect-intersections x y xx yy


end

to connect-intersections [x1 y1 x2 y2]
  ask patch x1 y1
  [
    ask intersections-here
    [
      create-redlink-to one-of intersections-on patch x2 y2
    ]
  ]
end
```

```
to make-intersection[x y]
  ask patch x y
  [
    set intersection? true
    set pcolor green
    sprout-intersections 1
  ]
end

to setup-roads
  ask patches [set direction? false]
  ask intersections
  [
    let x pxcor
    let y pycor
    ask out-link-neighbors
    [
      draw-road x y pxcor pycor
    ]
  ]
end

to draw-road [x1 y1 x2 y2];[here to there] or from [patch to patch]
  let numer (y2 - y1)
  let denom (x2 - x1)
  let g 1
  set g (GCD numer denom)
  set numer (numer / g)
  set denom (denom / g)

  if denom != 0 or numer != 0;if intersection is not on the same square
  [
    let ver-dir ""
    let hor-dir ""

    ifelse numer > 0 [set ver-dir "up"][set ver-dir "dn"]
    ifelse denom > 0 [set hor-dir "rt"][set hor-dir "lt"]

    let lastx x1
    let lasty y1

    while [lastx != x2 or lasty != y2]
    [
```

```
    ask patches with [pxcor <= max list lastx (lastx + denom) and pxcor >= min list lastx (lastx +
denom) and pycor = lasty ];cordinates in the range i want, which is the last patch to the patch
xdistance away horzo r vert
    [
      set direction hor-dir
      set direction? true
      set pcolor grey

    ]
    set lastx (lastx + denom);remove form loop
    ask patches with [pycor <= max list lasty (lasty + numer) and pycor >= min list lasty (lasty +
numer) and pxcor = lastx ];cordinates in the range i want, which is the last patch to the patch
xdistance away horzo r vert
    [
      set direction ver-dir
      set direction? true
      set pcolor grey

    ]
      set lasty (lasty + numer)
  ]



  ask patches with[pxcor = x1 and pycor = y1][set pcolor green]
  ask patches with[pxcor = x2 and pycor = y2][set direction? false]
 ]
end


to-report GCD [n d]
 if n = 0 or d = 0 [report 1]
 let num1 (abs n)
 loop
  [
   if(n mod num1 = 0) and (d mod num1 = 0)
   [
     report num1
   ]
   set num1 (num1 - 1)
  ]
 report 1
end
```

```
;---------End of Setup COmmands Only Run Commands Now-------------------------
;stoplights:



to work-intersections;i think this can be removed
  ask intersections
  [
    let r false
    let l false
    let u false
    let d false
    ask  patch-at 0 1
    [
      if direction? = true [ set u true]
    ]
    ask  patch-at 0 -1
    [
      if direction? = true [set d true]
    ]
    ask  patch-at 1 0
    [
      if direction? = true [set r true]
    ]
    ask  patch-at 0 -1
    [
      if direction? = true [set l true]
    ]
    set up? u
    set down? d
    set right? r
    set left? l

  ]

end


to work-stoplights;switch from vert to horizontal
  ask stoplights
  [
    set maxswitch stoplight-cycle-length
    if switch <= 0
    [
      ifelse vert? = true [switch-horizontal xcor ycor set switch maxswitch]
```

```
      [switch-vertical xcor ycor set switch maxswitch]
      ]


    set switch (switch - 1)
  ]

end
to switch-stoplight [sl]
  ask sl [
    ifelse vert? = true [switch-horizontal xcor ycor]
           [switch-vertical xcor ycor]
  ]
end

to switch-horizontal [x y]
  ask patch (x - 2) (y - 1) [set direction "rt"]
  ask patch (x + 2) (y + 1) [set direction "lt"]
  ask patch (x - 1) (y + 2) [set direction "st"]
  ask patch (x + 1) (y - 2) [set direction "st"]
  ask patch x y [set pcolor green]
  ask stoplights-on patch x y [set vert? false]
end
to switch-vertical [x y]
  ask patch (x - 2) (y - 1) [set direction "st"]
  ask patch (x + 2) (y + 1) [set direction "st"]
  ask patch (x - 1) (y + 2) [set direction "dn"]
  ask patch (x + 1) (y - 2) [set direction "up"]
  ask patch x y [set pcolor red]
  ask stoplights-on patch x y [set vert? true]
end


to make-stoplights [h w trx try];height, width, top right x cor, top right y cor **not in use
  ask patch trx try [sprout-stoplights 1]
  ask patch trx (try - (h - 1)) [sprout-stoplights 1]
  ask patch (trx - (w - 1)) try [sprout-stoplights 1]
  ask patch (trx - (w - 1)) (try - (h - 1)) [sprout-stoplights 1]

  ask stoplights[ask patch-at 0 0 [set direction? true set direction "fd"] ]

end


;--------------end of s tolights
```

```
;move, spawn, kill and cars stuff


to spawn-cars
  ask patches with[intersection? and pcolor = green]
    [
      if random 100 < spawn-percentage and not any? cars-on patch pxcor pycor
      [
        let dir direction
        sprout-cars 1
        [
          set shape "car"
          set color blue
          set go-dist 0
          ifelse random 100 < fast-car-spawn-rate [set maxspeed 2 set acceleration 1 set color red
][set maxspeed 1 set acceleration 1]
          set follow-dist car-follow-distance
          set waiting? false
          set tics-alive 0

          if dir = "rt" [facexy (xcor + 1) ycor]
          if dir = "lt" [facexy (xcor - 1) ycor ]
          if dir = "up" [facexy xcor (ycor + 1)]
          if dir = "dn" [facexy xcor (ycor - 1) ]
        ]
      ]
    ]

end

to move-cars
  let dir ""
  ask cars
  [

    ;let i 0
    ask patch-at 0 0 [set dir direction]
    if dir = "st" [set waiting? true set go-dist 0]

    if not waiting?
    [ let w? false
    ask cars-on patch-ahead 1 [if waiting?[set w? waiting?]]
    set waiting? w? ]
    ;while [i < spots and dir != "st"];stop
    ;[
```

```
    if go-dist < maxspeed [set go-dist (go-dist + acceleration)];used if cars accelerate
    if dir = "rt" and check-ahead follow-dist who and not any? cars-on patch-ahead 1;right
    [facexy (xcor + 1) ycor fd (go-dist * car-speed-multiplier) set waiting? false]
    if dir = "lt" and check-ahead follow-dist who and not any? cars-on patch-ahead 1;left
    [facexy (xcor - 1) ycor fd (go-dist * car-speed-multiplier) set waiting? false]
    if dir = "up" and check-ahead follow-dist who and not any? cars-on patch-ahead 1;up
    [facexy xcor (ycor + 1) fd (go-dist * car-speed-multiplier) set waiting? false]
    if dir = "dn" and check-ahead follow-dist who and not any? cars-on patch-ahead 1;down
    [facexy xcor (ycor - 1) fd (go-dist * car-speed-multiplier) set waiting? false]

    if dir = "fd" [fd go-dist * car-speed-multiplier] ;forward
    ;set i i + 1
    ;]


  ]
end

to turn-cars
  ask intersections
  [
    ask patch-at -1 -1 [ask cars-here []]
  ]

end

to-report check-ahead [x w];[how far ahead to check, the who # of the car]
  let move true
  let d x
  ask car w
  [
    while [d > 1]
    [
      if any? cars-on patch-ahead d  [ask cars-on patch-ahead d [set move waiting?]]
      set d (d - 1)
    ]
  ]
  report move

end

to-report check-right [x w]

end
```

```
to-report check-left [x w]

end

to despawn-cars
  ask cars
  [
    ask patch-at 0 0
    [
      if not direction?
      [
        ask cars-here [
          if collect-trip-length? [  set total-trip-length (total-trip-length + tics-alive ) ]
          die]

        if collect-trip-length? [  set num-of-cars-despawned (num-of-cars-despawned + 1)  ]

      ]
    ]
  ]
end

;---monitor stuff and adjusting variable

to select-stoplight
  if mouse-down?
  [
    let xmouse mouse-xcor
    let ymouse mouse-ycor
    if any? stoplights-on patch xmouse ymouse
    [
      if selected-stoplight != 0[unlabel-current]
      set selected-stoplight one-of stoplights-on patch xmouse ymouse ;change selscted stolight
      label-current
    ]
  ]
end
;; label the current light
to label-current
  ask selected-stoplight
  [
    ask patch-at -2 5
    [
      set plabel-color black
      set plabel "current"
```

```
    ]
  ]
end
;; unlabel the current light (because we've chosen a new one)
to unlabel-current
  ask selected-stoplight
  [
    ask patch-at -2 5
    [
      set plabel ""
    ]
  ]
end
;-------------------------
to display-timing
  ask stoplights
  [
    let sw switch
    ask patch-at -2 3
    [
      set plabel-color black
      set plabel (word sw)
    ]
  ]
end
to change-timing
  ask selected-stoplight [
    set switch input-switch
    let sw switch
    ask patch-at -2 3
    [
      set plabel-color black
      set plabel (word sw)
    ]
  ]
end

to monitor-cars-waiting

  let summ 0
  ask cars[ if waiting?[set summ (summ + 1)]]
  set cars-waiting summ

 ;if ticks mod 1000 = 0 [set CWS 0 set num-of-sums 0]
  ;set CWS (CWS + cars-waiting) set num-of-sums (num-of-sums + 1)
```

```
end
to reset-follow-distance
  ask cars[set follow-dist car-follow-distance]
end

to syncronize-lights
  ask stoplights
  [
    switch-horizontal xcor ycor
    set maxswitch stoplight-cycle-length
    set switch maxswitch
    display-timing
  ]
end
to randomize-lights
  ask stoplights
  [
    ifelse random 2 > 0[switch-horizontal xcor ycor][switch-vertical xcor ycor]
    set maxswitch stoplight-cycle-length
    set switch (random maxswitch)
    display-timing
  ]
end
to cascade-lights;only works if maxswitch is >=40, prioritize down and right traffic
  ;initializei it
  ask stoplights [switch-vertical xcor ycor]
  ask stoplights-on patch -40 40 [set switch 0]
  ask stoplights-on patch -40 0 [set switch 40]
  ask stoplights-on patch -40 -40 [set switch 80]

  ask stoplights-on patch 0 40 [set switch 40]
  ask stoplights-on patch 0 0 [set switch 80]
  ask stoplights-on patch 0 -40 [set switch 120]

  ask stoplights-on patch 40 40 [set switch 80]
  ask stoplights-on patch 40 0 [set switch 120]
  ask stoplights-on patch 40 -40 [set switch 160]


  display-timing

end
```