Hunter Loen
Dorian Dixon
Mike Jacquez

Stuck in Traffic

Decreasing Congestion Through More Efficient Traffic Lights

Table of Contents

Executive Summary

This program set out to tackle the problem of inefficient traffic lights. If a traffic light at an intersection can be simulated, the optimal cycle time for it can be found. Our result is a NetLogo program that simulates traffic flow at a simple intersection. One of the ways we found the best traffic flow was to raise the speed limit. Based on the findings from our program, we recommend abolishing speed limits.

Introduction

The inspiration for this project came, predictably enough, when one of the team members was wasting precious minutes of his life at a traffic light. From there, it progressed into a program that that simulates this common scenario, but without having to subject precious real humans to the inhumane stresses of a thick commute.

Purpose

The purpose of this program is to efficiently run simulations of a traffic light at an intersection so as to find the optimal light cycle pattern as far as traffic flow is concerned. The premise behind this program is that this will allow more cars to get through, increasing quality of life for all involved. Also, fewer cars idling means less fuel burned and, less smog created, and less greenhouse gasses released.

Significance

The significance of this project is obvious to anyone who commutes to work in a congested area. Cars were created to go places fast, not stay still. So, minimizing how much time they waste at traffic lights is a worthy cause in itself. There are also a host of environmental effects of traffic congestion.

Background

Early automobile drivers found that, if you have cars coming from four directions and all meeting up in one intersection, they will crash. This observation quickly led to the semaphore, which resembled a railway symbol. Around the beginning of the 20th Century, the first electric red-green traffic lights came into being. The first automatic traffic system was found in Houston in 1922 (ITE Traffic Control Systems Handbook). Traffic lights have not changed a lot since then, aside from optics upgrades and, of course, the volume of traffic.

Description

This program is written in NetLogo. It has the following variables:

- Speed limit
- Maximum acceleration cars are capable of
- Maximum braking cars are capable of
- What percentage of cars come from the north
- What percentage of cars come from the east
- How long the green light lasts
- How long the yellow light lasts.

The program simulates a simple intersection, where cars only go forward. It also only simulates traffic coming from the north and from the east. The user can manually control when the lights cycle or have it be automated.

Scope

The scope of this program extends to anywhere there are roads, cars, and traffic lights. Any commuter can see the value of maximizing traffic flows. This project goes beyond how long office workers spend in traffic, though. It also has environmental aspects, as when cars are sitting around idling, they release exhaust, which contributes to smog, which in turn contributes to asthma and lung problems, not to mention all the CO_2 emitted from these cars, which contributes to global warming.

Materials

All that is needed for this program to work is a computer running Windows 98 or higher (we used XP) with a modest amount of RAM, only about 256 MB. If this program were to be kicked up a notch, and made to simulate hundreds or thousands of different scenarios, it would require a bigger computer.

Methods

We employed the method of agent-based modeling for this program. This is the primary method used by NetLogo, which is the language/program we modeled this in. In agent-based modeling, each agent (in this case, cars) directs itself, as opposed to being centrally-controlled. Agent-based modeling works well for this situation because in real life cars are controlled by independent people, and not centrally controlled.

Computational Science Process

Our computational science process consists of gathering data from our simulation and interpreting it. We do not as of yet have a systematic approach. Our experiments mainly consist of tinkering with the variable to see what can be done to maximize traffic flow. This is how we discovered that higher speed limits are good for traffic flow.

Mathematical Model

This is not a scale model. We used speed based on the scale used in NetLogo, not an approximation of real speeds encountered on roads.

Computational Model

Our computational model revolves around creating as close to real-life conditions in the simulation, and then having a computer simulate their logical progression. It is helpful to use a computer in this, as they are much more systematic, have more stamina, and can simulate real-life with a fraction of the resources it would take to recreate the situations we are testing.

Appendix A
Program source code

```
globals [ stop-light ]
turtles-own [ speed ]
patches-own [ clear-in ]


;;SETUP PROCEDURES


to setup
  clear-all
  set-default-shape turtles "car"
  ask patches
    [ set pcolor brown
         if abs pxcor <= 1 or abs pycor <= 1
        [ set pcolor black ]
    ]
  set stop-light "north"
  draw-stop-light

end

to draw-stop-light
  ask patches with [abs pxcor <= 1 and abs pycor <= 1]
    [ set pcolor black ]
  ifelse stop-light = "north"
  [ ask patch 0 -1 [ set pcolor red ]
    ask patch -1 0 [ set pcolor green ]
  ]
  [ ask patch 0 -1 [ set pcolor green ]
    ask patch -1 0 [ set pcolor red ]
  ]
end


;;RUNTIME PROCEDURES


to go
  move-cars
  make-new-cars
  if auto? ; switch the light automatically
```

```
  [ if ticks mod (green-length + yellow-length) = 0
    [ switch ]
    if ticks mod (green-length + yellow-length) > green-length
    [ ask patches with [pcolor = green]
      [ set pcolor yellow ]
    ]
  ]
  tick

end

to make-new-cars
  if (random-float 100 < freq-north) and not any? turtles-on patch 0 min-pycor
    [
      crt 1
        [ set ycor min-pycor
          set heading 0
          set color 5 + 10 * random 14
          set speed min (list clear-ahead speed-limit)
        ]
    ]
  if (random-float 100 < freq-east) and not any? turtles-on patch min-pxcor 0
    [
      crt 1
        [ set xcor min-pxcor
          set heading 90
          set color 5 + 10 * random 14
          set speed min (list clear-ahead speed-limit)
        ]
    ]
end

to move-cars
  ask turtles [ move ]
end

to move ;; turtle procedure
  let clear-to clear-ahead
  ifelse clear-to > speed
  [ if speed < speed-limit
    [ set speed speed + min (list max-accel (clear-to - 1 - speed)) ] ; accelerate
    if speed > speed-limit
    [ set speed speed-limit ] ; but don't speed
  ]
  [ set speed speed - min (list max-brake (speed - (clear-to - 1))) ; brake
    if speed < 0 [ set speed 0 ]
```

```
  ]
  repeat speed ; move ahead the correct amount
  [
    fd 1
    if not can-move? 1
    [ die ]

  ]
end


to-report clear-ahead ;turtle procedure
  let n 1
  repeat max-accel + speed  ; look ahead the number of patches that could be travelled
  [ if (n * dx + pxcor <= max-pxcor) and (n * dy + pycor <= max-pycor)
    [ if([pcolor] of patch-ahead n = red) or
        ([pcolor] of patch-ahead n = orange) or
        (any? turtles-on patch-ahead n)
      [ report n ]
      set n n + 1
    ]
  ]
  report n
end


to switch
  ifelse stop-light = "north"
  [ set stop-light "east"
    draw-stop-light
  ]
  [ set stop-light "north"
    draw-stop-light
  ]
end
```