Learning Traffic Light Simulation

Lynn Jepsen
Computer Systems Lab, 2006-2007

June 12, 2007

Abstract

This project is meant to simulate a busy traffic light. The program recognize patterns in the intersection and then uses that information to make the light as efficient as possible. These patterns could be related to the time of day and/or the day of the week. At first I would purposely input recognizable patterns and see if the program would catch it, but eventually the plan would be to possibly use this program at a real intersection. There are many variables that the program takes into account including traffic density and number of lanes.
Keywords: traffic simulation, efficiency, cars, red light: stop, green light: go, pedal to the metal, voom voom, honk

1 Introduction

I want to develop an algorithm that will increase efficiency by changing the lights at the right time. I developed an algorithm that would decide when to make each direction red, yellow, or green based on information it gathers from the intersection. In order to see if this algorithm is working, and gather lots of data very quickly, I built a simulation of a traffic intersection to test the algorithm on. The simulation follows basic rules of the road. Each car takes up so much space on the lane, travels so fast, and has a max acceleration. It is a realistic simulation because it obeys physical laws as well as human laws. People only travel so fast and do not peel out of intersections. The data gathered from the intersection is then fed into the light algorithm, which changes the light in each direction.

2 Background

Traffic lights can cause problems with traffic flow if they are not timed well. If the light has no information about the intersection it is controlling, then you can sit at a red light for what seems hours. Even worse is when the light is much too short and you have to wait through lots of cycles. We have all experienced this. There has been a lot of research done to try and automate cars, using GPS, so that they all pass through the intersection harmlessly. However, this technology can be expensive, and it would take a long time to install it into all cars in order for the project to work. I think a much cheaper solution would be to fix existing traffic lights with more efficient algorithms. In order to see if this algorithm is working, and gather lots of data very quickly, I designed a simulation of a traffic intersection to test the algorithm on. This way I can test pictorially, the aerial view of the intersection, but also graph certain parameters such as queue length and wait time to see general trends.

3 Developments

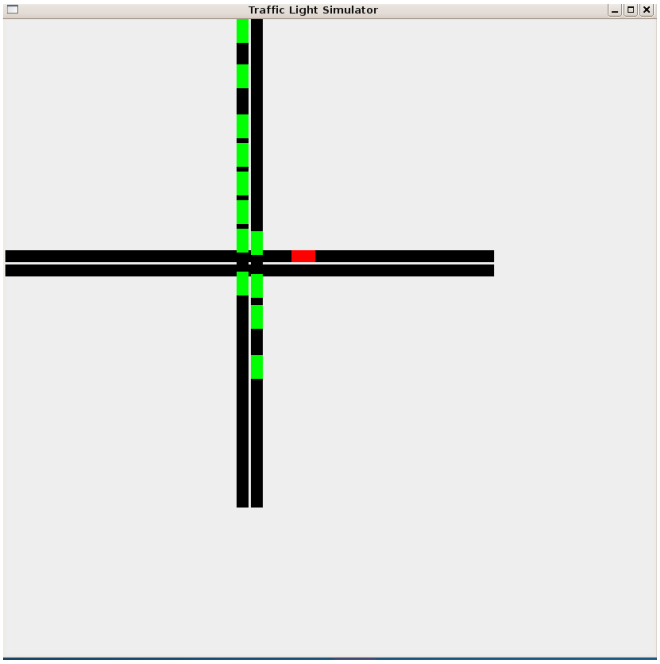
3.1 Hierarchy

In order to set up my program with appropriate an hierarchy to increase efficiency and make it easier to understand, I wrote four basic classes. The top class was the TrafficLight. It had my two most important functions: running the simulation, and the light algorithm. It also stored all four Directions. Direction was my next class. It stores all the lanes in a certain direction (north, south, east. or west) and can change the lanes of cars if the cars so desire. Each Direction stores a certain number of Lanes (depending on how big the user wants the intersection to be). Lanes have the job of telling cars to either stop or go through the intersection depending on whether the light is red, yellow, or green. The color of the light is passed from TrafficLight to Direction to each Lane individually. The Lanes also stores all the Cars, and makes sure they don't overlap. Cars store their speed and spot on the lane. They know how fast they want to go, how fast they like to accelerate, and other mechanical properties. Depending on the information the Lane gives it, the Car will either slow down or speed up each step of the simulation. The Car re-saves its current speed and spot on the Lane and then sends its

information back up to the Lanes class. All information goes back up to the TrafficLight, which runs the simulation and light algorithm

3.2 Simulation

The first section that I developed was a basic simulation of a four lane intersection. After programming so cars that move with realistic speeds and acceleration, stop behind each other, accelerate at reasonable speeds and obey other rules of the road, I then set each intersection at a set traffic density that can be changed at the beginning of each experiment. Traffic density is the number of cars then drive onto the road in a minute. The traffic densities are slightly random in that the average number of cars is close to the traffic density. For example, if the traffic density was 60, that doesn't necessarily mean that a car drives onto the road every second, just something close to that. This is a better representation of the real world. I later added multiple lanes, so I could test my algorithm on intersections made up of complex traffic densities and multiple lanes. The simulation is an easy way to test a light algorithm. If you look at the intersection and there are a lot of cars sitting somewhere and not moving, then perhaps the light algorithm could be doing a better job. It is not an exact science, but it certainly helped in the beginning, and it means more to people then any other graph. We all see intersections every day, so we know whats good and whats bad.



3.3 Light Algorithm

The next thing I worked on was the most important part, the light algorithm. The light algorithm is what decides when to change the lights from red to green in all four directions. In order to do this there are two variable that must be found. These are cycle length and ratio. Cycle length is how long it takes the intersection to go through an entire cycle with both sides having their chance to be green. Ratio is the ratio of green light time in the north/south direction compared to the east/west direction.

I wanted to find the correct variables for the light cycle at each instant on an intersection that would maximize efficiency. I defined efficiency with three other variables. They are queue length, wait time, and green light usage.[2] So if we consider cycle length and ratio to be our independent variables and our efficiency variables are our dependent variables, the experiment is to change th cycle length and ratio in order to optimize efficiency (all three

variables). However, each instant on the intersection has one other main variable that I cannot effect with my light algorithm. This is the traffic density. The light algorithm looks at previous information that has been stored from the simulation. It only looks at the past ten cycles because that is all the information it needs, and older cycles could skew the data. It then finds matches with the current traffic density (found by the Direction) and past cycles. Once it has those cycles in mind, it finds which cycles had the best queue length, wait time, and green light usage. This means it can find up to three separate cycles if the best of each variable is spread out. It then averages all three to find the best cycle for the intersection, at least at the moment. It then does exactly the same thing for the ratio component. This is the main function of the light algorithm, using previous information to make educated guesses about the current situation.

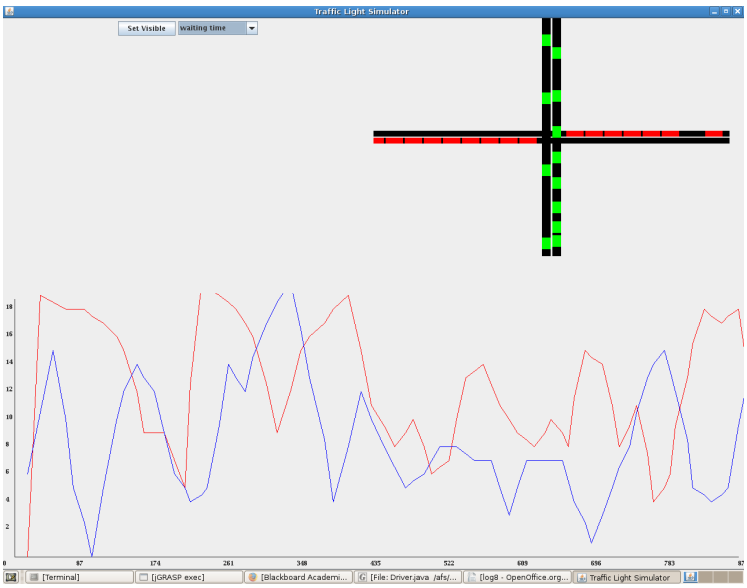
However, it has to somehow decide on cycles and ratios in the beginning without information. At first, the two functions find cycle length and ratio through other means. Cycle length simply took the max wait time in each direction and added them together plus ten seconds to account for red and yellow light time. I figured this would give enough time for lots of cars to get through the intersection. Ratio was decided by taking the queue length in each direction and dividing. These more basic functions are also used randomly through out the program to make sure that it doesn't get "stuck in a rut". If you end up using the same cycle over and over again, but the traffic pattern change, then you have nothing to look back on to change because it only looks back ten cycles. So about one in ten times the light algorithm uses these functions to decide on cycle and ratio instead of looking at previous information

The light algorithm will continue to compromise and eventually begin to use similar cycle lengths and ratios. It approaches an optimization after enough time. Considering that an intersection receives tons of data every day, I feel like an algorithm that needs lots of information is appropriate for the situation.

3.4 Graphs

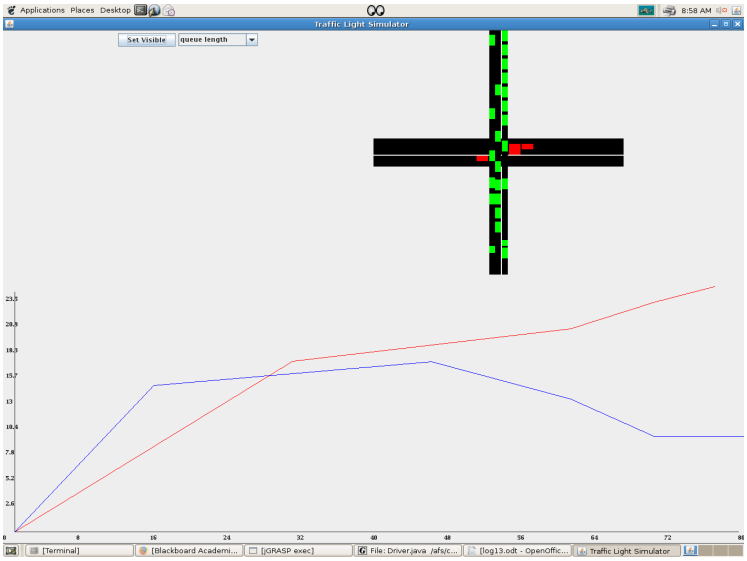
I added another visual component to my program once I had finished my light algorithm. It allows you to graph all three efficiency variables. This way you can see if the algorithm is really optimizing the intersection. I made is so that it graphed a north/south line and an east/west line. The closer

the two lines are to each other, the better the optimization. This is a more exact way of measuring efficiency than just looking at the simulation. Note that the intersection can only be so efficient. If there are just too many cars for the number of lanes, then the algorithm will not work as well as it would in a lighter traffic density. This is not an exact science. There are too many variables in effect here, and they are often very random due to the nature of the road. You cant possibly minimize wait time to the point where no one waits. Traffic flow is just too erratic to predict well.



3.5 Multiple Lanes

I updated my simulation to be able to include multiple lanes in any direction. While this does tend to lighten the traffic density, it has little other effect on the light algorithm. I did have to increase the yellow light time when there are too many lanes, because other wise too many cars run red lights. I have yet to make it so that cars can change lanes, but there is room in the hierarchy for it.



4 Results and Discussion

After running many simulations on heavy traffic flow, I have discovered that there are two possible ways to efficiently run a light. The first way is to have a cycle length that is just long enough to let all the cars through in a green light before switching. This cycle length is very hard to guess because of the randomness of traffic flow. The second way to run a light would be to have an incredibly short cycle length that only lets one or two cars through an intersection at a time. This way both directions are in effect moving at the same time instead of having to wait at a light for 20 seconds. Both methods tend to have about equal queue length, but long traffic light cycles have longer wait time and short traffic light cycles have worse green light usage. The wait time is shorter in the short traffic light cycles because you are constantly moving, even though you are stop and go. The wait time includes the entire time behind a light, so it is not just the wait time for one cycle. The reason why longer traffic light cycles have better green light usage is because they are designed to have exactly the right amount of time to let in all the cars in a direction. I believe that the longer traffic light cycles are a lot harder to make work right. Traffic densities tend to be too random to allow for a perfect calculation of how long each cycle should be. I also believe

that short traffic light cycles are probably, on the whole, less efficient because of the proportionately more yellow light time in each cycle. This is a factor I did not take into account in my program, but would have in hind sight. My conclusion is to have short traffic light cycles on intersection where the light algorithm is not very well equipped (doesn't have a lot of sensors) or if the intersection tends to be very random. On the other hand, if the intersection has lots of information, then longer traffic light cycles would work better to create a more efficient intersection.