Coordination of Traffic Signals in Rural Areas with Machine Learning

STEM THESIS

Shane Ferrante

Table of Contents:

Abstract

In most semi-rural areas, traffic lights all act independently without taking into account the status of other traffic lights, leading to a lack of coordination and an increase in the total wait times and travel times of cars in these areas. The aim of this project is to engineer and simulate a modular algorithm that controls traffic lights more effectively than the current method by coordinating them to reduce the average time that cars spend waiting at red lights. Currently, only three percent of traffic lights in the United States are considered "smart," meaning that they take into account the status of the cars and lights around them. These "smart" traffic lights are almost entirely concentrated in urban areas. Rural areas are often overlooked because the benefit of more advanced traffic systems is often underestimated. To show the superiority of a traffic system with coordinated lights over one with simple timers, a simulated model of a city/town was created in Java in which to test the systems. A genetic algorithm(a type of machine learning) was then used to train the traffic lights to coordinate with each other. Then, the two systems were compared by running several tests over different city sizes and traffic densities, and it was shown that, on average, cars in the system with coordinated lights spend up to 21.5% less time at red lights than those in the simpler system. Thus, rural regions would greatly benefit from a system of coordinated traffic lights.

Acknowledgements

This project was supported and helped by many people who guided me along the way. Ms. Small and Ms. Taricco were my two main stem advisors throughout the project and provided various help with many different things along the way. Mr. Ellis and Mr. Regele also helped as co-stem advisors during a part of the project as well. Dr. Crowthers has also been a tremendous help with all that he has done from the start of the year in all aspects of the project. My parents have been huge emotional supporters throughout the whole project and have kept me on track. Lastly, a few of my peers have been major helps as well. Vinnie Jeyakumar, Arjun Iyer, Matthew Young, Pragya Narahari, and Toyesh Jayaswal have all helped my in my project emotionally or by guiding me with advice on how to improve and continue.

Literature Review

Introduction

Traffic is a heavily researched field because of its importance in the everyday life of almost everyone. One major subsection of this research is the optimization of traffic signals. All drivers rely on traffic lights to work consistently and efficiently to reach their destination as quickly and safely as possible. There is immense variation in the way traffic signals operate from city to city and even from intersection to intersection. Some have sensors and cameras that give the signal information about where cars are coming from, some can communicate with surrounding lights to more effectively direct traffic flow, and some operate solely based on a timer which tells the light how long to be green and how long to be red. It turns out that only three percent of all traffic lights in the United States are considered "smart" (Austin, 2019). The others have a simple timer which controls them with no knowledge about their environment or surrounding lights. This results in wasted time across the United States. However, with smarter traffic lights, travel times can be greatly reduced. Both sensors and coordination of lights have been shown to decrease travel times across the board. However, most of the time, smart lights are more expensive and more difficult to implement than a standard timer. Pressure plates, induction loops, and cameras can cost a lot of money to implement across a whole city. Coordinating lights, however, is a mostly software-based solution, and thus potentially cheaper. Traffic lights can be coordinated in many ways, such as making two adjacent lights work in tandem or programming a whole city to work together. There is a lot of variation between lights because different areas require different types of traffic management. Extremely dense urban centers require advanced sensors and coordination, whereas towns usually do not require as much technology. This is generally why more rural areas have more simple lights because there is not as much of a need for them as urban areas. Because of this, rural areas are often swept under the rug when it comes to researching better traffic management methods.

One popular method of controlling and coordinating lights which is heavily researched but rarely implemented is machine learning. There have been many articles published that deal with using machine learning to coordinate traffic lights in numerous ways (Bazzan, 2005) (Dresner & Stone, 2005) (Gregoire, Desjardins, Laumonier, & Chaib-Draa, 2007) (Tonguz, Viriyasitavat, & Bai, 2009). To do this, they must simulate a small array of traffic lights and roads to see how well their algorithm performs. Extensive research has been done on different methods to control and coordinate lights, different ways to simulate traffic, and different ways to measure the effectiveness of a specific strategy. However, there is a lot of disagreement on what is the best way to do each of these. These disagreements make it difficult to determine whether one method of coordinating lights is truly better than another. Thus, these algorithms are rarely implemented because of this disagreement.

Traffic Lights

There are many aspects of traffic lights that all contribute to its functionality and keep drivers safe on the roads. The most basic form of a traffic light is that of a four-way intersection with traffic coming in and out of all directions. In this case, the lights alternate between the two perpendicular directions. However, even this is not quite as simple as it may seem. There are many aspects which must be optimized in a system like this to ensure the safest and most efficient travel for everyone. First, the cycle length must be adjusted based on the conditions of the intersection to ensure that it is not so long that people in other directions are waiting at a red light for no reason and not so short that some of the backed up cars have to wait two or three light cycles to get through a single intersection. Secondly, if one direction is busier than the other then the green split (the division of green time between two directions) must be adjusted to give more time for green in that direction (Tonguz et al., 2009). There is also an intergreen period for traffic lights in which none of the lights are green to avoid one lane starting

as traffic is still clearing out of the other (Moor, 2019). This includes the time in which the light is yellow as well as a short period of time where all lights are red. Lastly, there must be a built in function to make all lights red to let crosswalks open up so that pedestrians can get through the intersection. Each of these aspects of a light are fine-tuned based on the conditions of the intersection. For example, the city of Boston has specific guidelines which tell how traffic must be analyzed to optimize the control of traffic lights (Boston, 2018). This also includes data on different times of day, so that lights will act much differently at during peak travel times like 7:00-9:00am and 4:00-6:00am compared to 2:00am for example (WSDOT, 2019). Even for a simple four-way intersection, traffic lights can become quite complicated. This is enhanced greatly when the intersections themselves get more complicated with multiple lanes, left turn only lanes, and right turns on red. With these additions, there is no longer only two phases which must be considered but many more including time when only left turns can be made, or only one of the four directions may proceed. For this reason, urban traffic is studied and monitored the most because large cities require these complicated traffic lights to deal with mass amounts of people traveling every day.

Methods of Reducing Traffic

Since traffic is such a big problem in many parts of the United States as well as other parts of the world, there have been many attempts to reduce traffic in various ways through different means. One example of this is physical sensors which detect vehicles near a traffic light to inform the traffic light about whether changing from red to green is even necessary. There are a few different forms of physical sensors existent across the US. The first of these is cameras which detect cars with image recognition software to tell the signal where cars are and where they are not. Another form of a physical sensor is a pressure plate or induction loop which is underneath the ground and detects the

pressure of a car on the road. Cameras, pressure plates, and induction loops effectively perform the same task of detecting the presence of cars and instructing the light to act accordingly.

Another important means of reducing traffic in the US is coordinating lights. There are many ways to do this, and each of them has its advantages and disadvantages. The first method is to coordinate lights on busy streets so that the people moving along them will be able to get through multiple lights at a time before needing to stop. Cambridge traffic lights work like this by coordinating intersections along arterial corridors to manage traffic effectively (Cambridge, 2006). The benefit of this method of coordination is that lights can work in conjunction so that all the parallel traffic may pass in one stage, and all the perpendicular traffic may pass in another stage. In this way, the lights parallel to the busy street will have a longer duration of green so that more cars will be able to get through multiple intersections in only one stage. However, there are a few problems with a system like this. First, perpendicular traffic flow is greatly slowed by this strategy because they must wait at longer red lights, and they have relatively short green lights. In some cases, cars may need to wait for up to two whole minutes at one light (WSDOT, 2019). One way to combat this is with green wave. Green wave is a strategy in which traffic signals along one street are coordinated, but slightly offset so that the light will only turn green when traffic from the other intersection reaches the light. To do this, the traffic lights need to be timed well, and it is much less forgiving than the other method. On the plus side, the perpendicular traffic does not need to wait nearly as long. Another problem with strictly timed coordination arises when crosswalks start being considered. Crosswalks are an important part of every intersection, but they complicate a traffic light's ability to function efficiently. The unpredictable nature of when pedestrians will want to cross or not makes strict coordination almost impossible by itself. This is because if one intersection along a coordinated street is delayed for twenty seconds by a crosswalk, then it will now be offset from the others by twenty seconds. This can potentially produce the exact opposite of the intended effect, and slow down traffic rather than speed it up. To combat this,

coordinated signals must also have a method of re-aligning with the other lights so that they can resume coordination. This often means that a light will need to wait for the rest to catch up so that everything returns to normal. Despite its many flaws, if fine-tuned to a specific location and done properly, one-dimensional coordination of lights along a street can be quite effective.

Urban and Rural Areas

Traffic in urban and rural areas are very different from one another, and likewise the means by which they are controlled also tend to vary drastically. In highly urban areas such as Los Angeles, New York, Boston and other major US cities, there is a high traffic density with complicated lights and multiple lanes on almost every street. This is a different environment from rural areas which have much simpler intersections, low traffic densities, and simple roads. Urban areas tend to also be more consistent for various reasons. First, it is likely that there is always going to be many cars on every street. Although this seems like a disadvantage, it means that traffic lights rarely need to worry about having a green light when there are no cars coming because chances are that there will always be cars coming from every direction. Because of this, sensors are not effective tools to use on their own because most of the time they will be detecting cars from all sides which provides little information or advantage to reducing traffic. However, coordinating lights is very effective in urban areas because keeping traffic flowing properly is necessary in an area with high traffic densities (Gregoire et al., 2007). This is because if one light is green for an extended amount of time, and the next light along that road is red, the traffic will potentially back up all the way to the first intersection causing many problems. This problem is unique to urban areas because in rural areas, traffic density is rarely high enough to have traffic back up that far. There is also another concern which is specific to urban areas the desire to steer people away from driving through residential areas. According to Cambridge's signal policy, one of the

main reasons for coordinating lights along arterial corridors is to deter drivers from taking shortcuts through residential areas (Cambridge, 2006). This is another problem which is unique to urban areas because in rural areas there is little worry about mass amounts of traffic being redirected through residential districts.

Rural regions are much different from urban areas, and they require different types of intersections and traffic lights to work effectively. In these areas, there is little worry about extreme backup due to the low traffic density, and for that reason there is not as much of a need to have advanced lights. Therefore, many rural areas still use default classic timers because timers are good enough for the job in most cases, and they get along without much trouble. However, that does not mean that coordinating traffic lights will not greatly reduce traffic. Instead, it would be beneficial to these regions and would save a lot of time over the current system if done properly. The only reason this is currently not implemented in many areas is because of the lack of necessity. Since there is no dire need for intelligent lights, then they are often overlooked. There is also one last type of region which are areas with extremely low traffic densities. This not only includes secluded rural areas, but also standard towns and cities at nighttime when traffic is low. In this case, physical sensors work their best because it is much more likely that a light will detect traffic coming from one direction but not the other.

Machine Learning

Machine learning is a useful tool in coordinating traffic signals, and it has been used in many papers on potential ways to coordinate lights. For example, as used in "A Distributed Approach for Coordination of Traffic Signals" (Bazzan, 2005), an evolutionary genetic algorithm may be used to coordinate lights two-dimensionally. This means that not only the traffic lights on one road are coordinated, but a whole network of lights is coordinated. Although this may slightly increase wait

times on certain roads compared to if they were coordinated one-dimensionally, the global average wait time will be lower meaning that on average, cars will spend more time moving and less time waiting. Two-dimensional coordination, or coordination of a whole network of lights is much more complicated than the other forms of coordination mentioned earlier. Thus, they cannot be implemented and tweaked manually by people because there are too many factors to consider. Therefore, machine learning is necessary because it is an effective way to tweak all these factors and adjust them to create a product which optimally coordinates traffic signals. One thing that is necessary when using machine learning is a way to measure how effectively a system of controlling traffic lights is working. This can sometimes be difficult to pinpoint because it is hard to prove that one method is explicitly better than another. One way to measure the effectiveness of a system is to compute the average wait time at a traffic light per car using the formula below.

$$\frac{1}{|C|} \sum_{v_i \in C} t(i) - t_0(i)$$

Figure 1: An equation to compute the fitness of a model

This formula (Dresner & Stone, 2005) is a simple and effective way to determine how well an algorithm controls a set of lights. The way it works is by taking each car(v_i) and subtracting the actual time it takes to get through its route(t(i)) by the theoretical time it would take if every light was green(t_0(i)). This gives the total wait time at all lights, then if that number is divided by the total number of cars(|C|), it will yield the average wait time of all cars at red lights.

There are many programming languages in which machine learning can be implemented, as well as many different types of machine learning each with various use-cases, advantages, and disadvantages. Python, MATLAB, and Java are all great languages that can handle machine learning with

various pros and cons. One pro about Python is that it is useful for machine learning with programs like TensorFlow. MATLAB is also known for its machine learning capabilities, and these two languages are known for their superiority in deep learning and reinforcement learning. Java, on the other hand, is not as advanced in this category because the language does not focus on fast low-level computations. However, there are still some machine learning libraries for Java such as Weka (Witten et al., 1999) and Java-ML (Abeel, Thomas, & Yvan, 1970) which can perform these same tasks.

Conclusion

Extensive work has been done in the field of making traffic signals more efficient and adapted to their environments. This includes the use of sensors to detect where cars are on the road and coordinating traffic lights to work together to improve traffic flow. Especially in urban areas, these advancements have been put to great use, and they are vital to the everyday workflow of people in America. However, rural areas have not experienced the same level of research, and there is a lack of work being done to alleviate traffic specifically in these areas. Because of this, rural areas tend to be stuck with traffic signals which do their job, but there is a lot of potential efficiency which is lost because of the simplicity of these traffic lights.

Introduction

In most semi-rural areas, traffic lights all act independently without taking into account the status of other traffic lights. This leads to a lack of coordination and an increase in the total wait times and travel times of cars in these areas. Coordinated lights would also be able to reduce the time cars are stopped at lights. Most studies on coordination of traffic lights focus on urban areas, and rural areas are often overlooked. Currently, only about three percent of all traffic lights in the United States are considered "smart." These traffic lights are almost entirely found in urban areas because of their high cost. A lot of these lights are pre-timed based on the time of day through estimated traffic data in those areas (Austin, 2019). Some smarter lights have sensors that detect cars and can simply switch the light from red to green if there is a car waiting for no reason. These sensors work effectively when there is no traffic coming from the other direction, but the problem with these is that they will only be useful in areas with very low traffic densities or at night. This is because these sensors are completely useless unless there is no traffic coming from the other direction. On the other side of the spectrum, large urban areas often will have coordinated traffic lights, but their function is mainly to minimize backup between intersections rather than to keep traffic moving steadily (Cambridge Traffic, 2006). For instance, the traffic control signal policy in Cambridge MA states that intersections along a busy street will be coordinated in order to reduce delays to discourage cutting through residential neighborhoods (Cambridge Traffic, 2006). All of these systems work well in their environments, but there is little use of these techniques in areas with traffic which is high enough that there is almost always a car in every intersection, but low enough that traffic never gets backed up too far. With the current traffic system, people in rural areas are potentially losing a lot of time waiting at traffic lights which can be avoided. According to Boston's traffic signal operations design guidelines, there is no mandate to coordinate traffic lights (Boston Transportation Department, 2018). Because of this, most towns simply use pre-timed lights which waste time.

The objective of this project is to create and simulate a method for controlling traffic lights for rural areas which decreases the amount of time cars spend waiting at red lights. Since this project is software-based and simulated, there are a few other aspects of the project to take into account. The first aspect of the project is how to effectively model traffic in a city or a town. There have been many studies before which have dealt with how to model traffic effectively in different settings (O. Tonguz & W.Viriyasitavat, 2009)(Dresner & Stone, 2005). For example, O. Tonguz and W. Viriyasitavat (2009) created a model of urban traffic using cellular automata. This means that a city was split up into cells large enough for exactly one car to fit in each cell. The advantage of this is that computing how cars move becomes much easier with this type of a system. Another aspect of my project is the implementation of green wave ("Traffic Signals", 2019). Every traffic light has a set amount of time which they will stay green and a set amount of time that they will stay red for any given time of day. However, in some smart traffic lights, this amount of time can be adjusted based on the traffic conditions or based on the traffic lights around them. This adjustment is called green wave and it is very important in many different traffic light coordination systems because it is a very effective tool for coordination.

The question about what makes one traffic system better than another is also very important to this project. For this, a fitness function which tells quantitatively how much better one traffic system is from another is crucial. To do this, there is a very effective formula from an article entitled Multiagent Traffic Management by K. Dresner (2005) which gives the average delay of a given traffic light for each car that goes through that light. All these aspects must be taken into account when creating my model and algorithm.

Methods/Materials

This project was broken down into three phases: modeling, creating the algorithm, and machine learning. Before beginning the process, the method of machine learning was decided. The three candidates were deep learning, reinforcement learning, and a genetic algorithm. A genetic algorithm was selected because of its prior success in other traffic light coordination algorithms. Plus, reinforcement learning and deep learning did not seem to fit the needs as well, and they would be much more convoluted to implement. All coding was done from scratch in Java in the Eclipse editor.

To model a simple city, the city is represented by a 2-dimensional rectangular network of traffic lights of a variable size. Cars are represented by points on the network and travel along roads from intersection to intersection. The model was modularly developed so that the size of the city, traffic density, traffic light duration, crosswalk frequency, and many other low-level variables could easily be modified. Time is represented as a tick-based system where each tick would correspond to approximately one second in the real world, but it is not meant to represent any specific unit of time in the real world. The functionality of the model is to run trials on different traffic light algorithms to test how well they perform. To find the optimal amount of time each trial should last, many trials were done on a simple city with varying lengths of time(in ticks) to measure both the consistency of the model at that time interval and the amount of real-time it took to run each trial. The goal of this is to maximize the consistency of the model while also keeping time manageably low.

To control the traffic lights, a modular algorithm is used which takes into account the status of the other lights around it. Each city uses one algorithm to control all of the traffic lights based on a three-dimensional array of parameters. Two of these dimensions denote which traffic light in the two-dimensional array is being controlled, and the third dimension represents each of the eight parameters that tell the traffic light how to operate. Each traffic light then uses these parameters along with

information about the surrounding traffic lights at any given moment to decide how to operate. Each light has a default duration, and this duration is adjusted on the fly based entirely on the current state of the surrounding environment and the parameters. To decide how well a specific algorithm controls the traffic lights of a city, a fitness function is used which measures the average wait time of each car at a red light during the trial (Dresner, 2005).

$$\frac{1}{|C|} \sum_{v_i \in C} t(i) - t_0(i)$$

Figure 1: An equation to compute the fitness of a model

This fitness function calculates the total time it would take for one car to go from point A to point B without any traffic lights theoretically, then subtracts that from the actual time it took which was measured by the model. Then it sums all of these differences and divides by the number of cars to obtain an average.

To train the model and optimize the algorithm, a genetic algorithm was used on a population of different traffic control algorithms. For clarity, the word “algorithm” is both used to describe a method in which a set of traffic lights is controlled as well as in the term “genetic algorithm” which is a form of machine learning. A genetic algorithm is a form of machine learning in which a population of organisms is created and tested to see how they perform at a given task, then a fitness function is used to determine the best organisms out of the population. These organisms then reproduce either sexually or asexually depending on the algorithm. This process is iterated until the organisms in the population are optimized. In this case, each organism is a traffic control algorithm which is represented by its parameters. The fitness function used is the one mentioned earlier which calculates the average wait time per car, and one trial is run for each algorithm. This process was repeated many times on a

population of 100 organisms until the algorithm was optimized enough that it could no longer improve considerably in a reasonable amount of time. Data from before optimizing and after optimizing was collected on many different city sizes as well as many different traffic densities (see results section).

Results

For the first prototype of my model, one major part of the design was how many ticks each trial should run for. To decide this, tests were run to determine how quickly the code could complete trials of different lengths. Trials were run on a 5x5 city of traffic lights with independent traffic light control algorithms. Fitness standard deviations were all calculated by the program and recorded after the trials. Time was recorded with a simple stopwatch.

Table 1: Preliminary timing data for determining optimal ticks/trial.

Ticks/Trial	# of Trials	Time #1	Time #2	Time #3	Fitness Stdev	Avg Time/Trial
		(seconds)	(seconds)	(seconds)	(ticks/car/light)	(seconds)
1000	**200**	6.57	6.63	6.54	0.177	0.033
2000	**100**	6.48	6.58	6.20	0.134	0.064
5000	**50**	8.08	8.25	8.58	0.082	0.166
7500	**50**	12.50	12.43	12.45	0.071	0.249
10000	**50**	16.02	16.48	16.57	0.071	0.327
25000	**20**	16.20	17.27	16.61	0.032	0.835
50000	**10**	17.10	17.13	17.35	0.031	1.719

This prototype was also used to get preliminary data on average fitnesses to get a ballpark for what to expect in future trials. For this, two very large trials were run on 7500 ticks per trial on a 5x5 city with independent traffic light control algorithms.

Table 2: Long trials for preliminary fitness data.

Ticks/Trial	# of Trials	Time	Average fit	Stdev
		(hours)	(ticks/car/light)	(ticks/car/light)
7500	**30000**	~1.5	4.8956	0.06997

7500	**100000**	~4	4.8963	0.06988

After creating the genetic algorithm, trials were run to evaluate how well the traffic lights were learning. This model of the genetic algorithm used single-parent creation of the next generation. Data was collected across 10 generations with 100 organisms per population. All tests were run on a 1x5 city with 100,000 ticks per trial and no mutation between generations. Each generation took approximately 40 seconds, and the average and best fitness values for each generation were recorded by the model.

Table 3: Prototype 1 fitness and genetic algorithm data

Generation	Avg. Fit	Best Fit
0	3.961	3.494
1	3.943	3.723
2	3.856	3.678
3	3.891	3.725
4	3.778	3.642
5	3.77	3.659
6	3.711	3.538
7	3.969	3.833
8	3.717	3.619
9	3.893	3.629

The second prototype for the genetic algorithm was tested similarly to the first. Single-parent creation was used again with 100 organisms per population on a 1x5 city at 100,000 ticks per trial and a mutation factor of 0.05. 101 generations were run and data was recorded only for the initial and final

few generations. Each trial took approximately 42 seconds on average, and all calculations were recorded and computed by the model.

Table 4: Prototype 2 fitness and genetic algorithm data

Generation	Avg. Fit	Best Fit	Worst Fit	STDEV
0	3.989	3.398	4.728	0.264
1	3.969	3.679	4.218	0.119
2	3.501	3.378	3.65	0.067
3	3.741	3.524	3.974	0.085
4	4.623	4.246	4.943	0.131
99	4.263	3.953	4.472	0.102
100	3.914	3.754	4.283	0.086
101	3.712	3.577	3.857	0.052

The final prototype for the genetic algorithm ran three sets of trials on how the learning process of the model was affected by different factors. The factors tested were city size, traffic density, and corridor length. Each trial was run on a 3x3 city with a default traffic density of 2 (unless otherwise specified by the trial). All populations contained 100 organisms which used a two-parent genetic algorithm at 100,000 ticks/trial and a mutation factor of 0.05. Trial lengths varied drastically based on city size and density. Graphs plot the average fitness over generations and the different lines represent different trials in the same set.

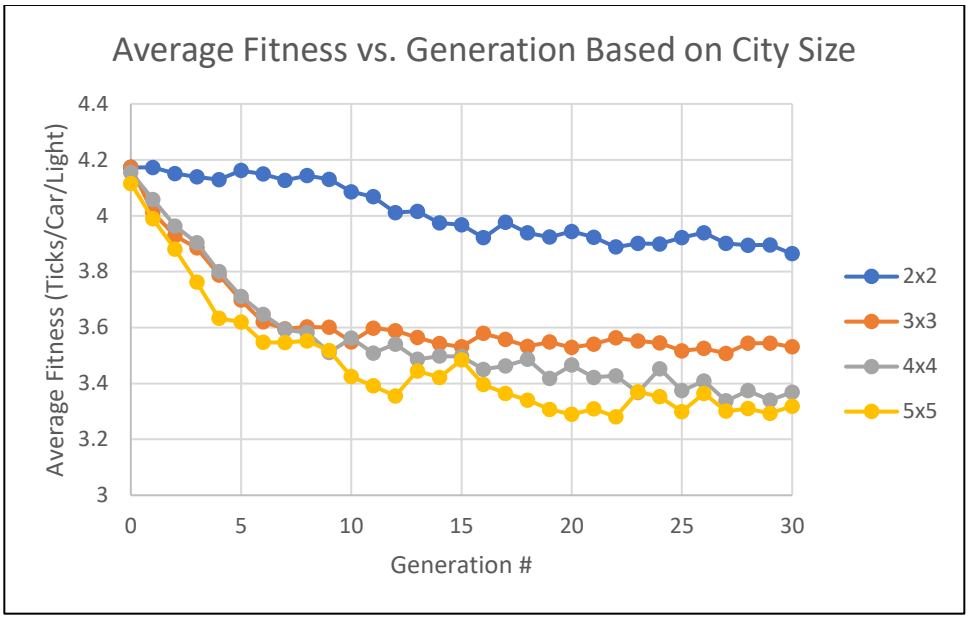

Figure 2: Average fitness vs. generation number for different city sizes.

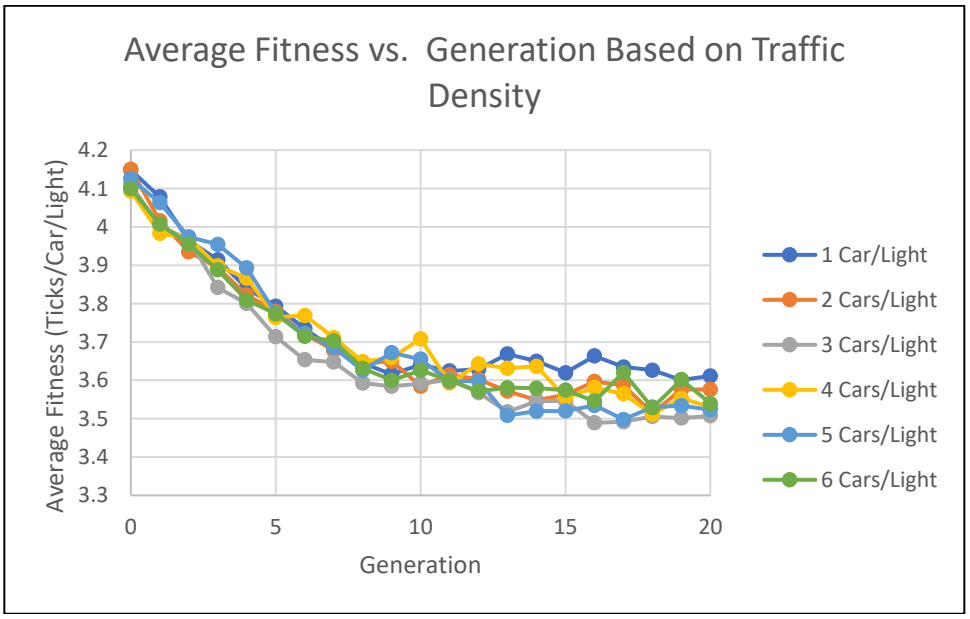

Figure 3: Average fitness vs. generation number for different traffic densities.

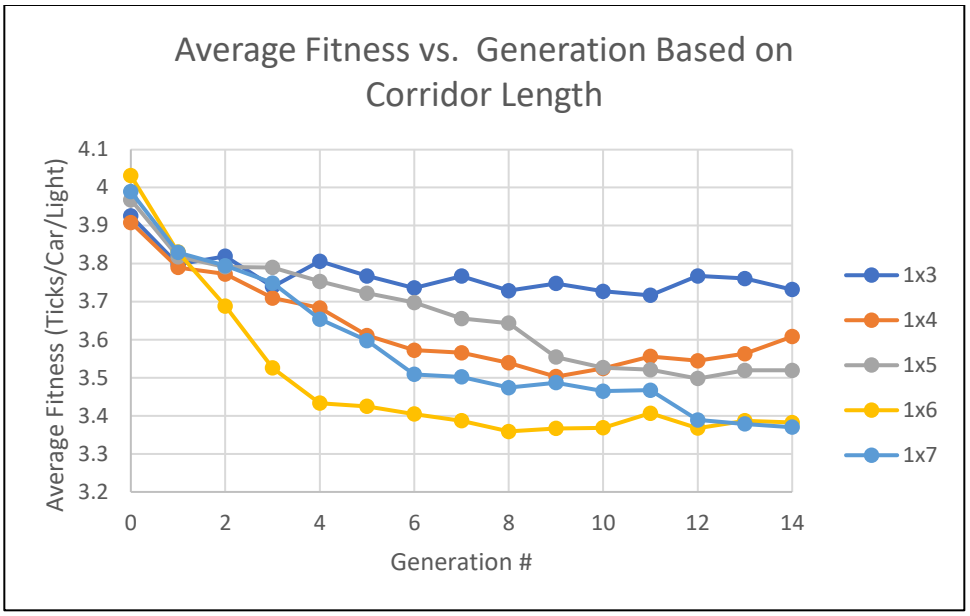

Figure 4: Average fitness vs. generation number for different corridor lengths in a 1 by n corridor.

For each of these trials above, the percent decrease in average fitness from start to finish was calculated and put into a table based on the trial run. P values were also calculated by a paired t-test from the first generation to the last to determine the statistical significance of the decline in fitness.

Table 5: Percent decrease in fitness based on city size.

City Size	Percent decrease	P-Value
2x2	8.41	7.78E-34
3x3	15.93	1.08E-48
4x4	19.68	7.66E-40
5x5	21.48	1.88E-60

Table 6: Percent decrease in fitness based on traffic density.

Traffic Density (Cars/Light)	Percent Difference	P-Value
1	14.37	4.94E-27
2	16.78	3.39E-29
3	15.96	1.33E-33
4	16.82	4.53E-40
5	16.09	4.36E-34
6	15.01	2.16E-25

Table 7: Percent decrease in fitness based on corridor length.

Corridor Length	Percent Decrease	P-Value
1x3	5.76	1.90E-05
1x4	10.37	1.22E-17
1x5	14.83	1.25E-26
1x6	17.94	3.90E-32
1x7	16.89	1.39E-33

Discussion/Conclusion

The objective of this project is to create a system which decreases the amount of time spent waiting at traffic lights. When analyzing the first prototype of the model, the main thing which was being varied as the number of ticks per trial i.e. the amount of time each trial ran. In theory, increasing the number of ticks per trial will increase the amount of time it takes to complete a trial, but the precision of the measurements made will be higher. This was shown to be the case because as the number of ticks was increased, the time it took to complete trials increased and the standard deviation of the fitness decreased. This means that there was less variation in the fitness measurements of models which ran longer. This prototype was also used to get preliminary data on average fitnesses, and the average fitness for a 5x5 city with independent lights came out to about 4.9 ticks/car/light. This means that on average, a car spent about 5 ticks at each traffic light.

To analyze the first prototype of the genetic algorithm, trials were run on generations of 100 organisms to track the learning of the model over time. The expected outcome of this testing was that the average and best fitnesses for each generation would decrease until it reached an optimal value. This is because the algorithm favors lower fitness because low fitness means that cars spend less time waiting at traffic lights. However, the average fitnesses seemed to go down for a few generations, then return right back to where they started and repeat as seen in table 3. In the second prototype, something very similar happened, but the difference was that this time standard deviation was tracked. This showed that the standard deviation of the fitnesses would always drop drastically from the first generation then remain fairly constant. This phenomenon was due to the fact that the first two prototypes used a genetic algorithm with asexual reproduction or one parent per offspring. The drawback of this is it drastically reduces variation in a population. Two-parent genetic algorithms, however, tend to have large variation initially and slowly converge on an optimal solution. Because of this, for the third and final prototype of my genetic algorithm, a two-parent system was used. This fixed

the issues, and the genetic algorithm worked as expected. After this, the average fitnesses gradually began to decrease as time went on until an optimal value was reached.

Three major sets of trials were run to collect data on the effectiveness of the coordination of traffic lights under different circumstances. These trials measured the effects with respect to city size, traffic density, and arterial corridor length. In the tests measuring city size, one clear trend was that the larger the city, i.e. the more traffic lights being coordinated, the better the algorithm performed (t-test performed on final generation fitness values from a 2x2 to a 5x5 city, $p=2.55E-46$). This is as expected because adding more lights to a coordinated system should theoretically improve performance. One conclusion which can be drawn from this is that coordination will have a significantly larger impact on big urban areas rather than small rural areas. However, this conclusion is somewhat misleading because of the fact that this trial assumed constant traffic density. A more logical conclusion would be that large rural areas would benefit more than small rural areas and likewise for urban areas. This conclusion is supported because in this trial traffic density was not changed, meaning that all areas were tested in a rural setting. The second major trial which was run varied traffic density while keeping city size constant. This trial showed that traffic density has a very minuscule effect on the fitness of the model when compared to other factors like city size. Although differences in traffic density do produce a statistically significant effect on the fitness of the model (t-test performed on final generation fitness values from cities of high and low traffic densities, $p=0.0113$), the benefit of coordinating areas of lower traffic density is similar to that of areas of higher traffic densities.

The final set of trials run varied the length of arterial corridors being coordinated. The model found longer corridors tend to benefit more from coordination than shorter corridors (t-test performed on final generation fitness values from corridors of long and short length, $p=3.14E-12$). All trials run showed vast improvement from independent lights which were extremely statistically significant with p-values mostly less than 1E-20. This type of traffic light coordination has been shown to decrease red-

light wait times by up to 21% in optimal circumstances. The main sources of error in this project would all come from the assumptions made in my model. These assumptions include the fact that all cars drive the same way and abide by all traffic laws, the fact that all cities are exactly rectangular, and that lights would theoretically be able to communicate with each other instantly, given the right equipment. In the future, the next step would be improving the model to take these factors into account to provide a more accurate representation of real-world traffic. This system provides a new way of looking at traffic from a rural perspective which is often extremely overlooked in the world today.

References

1. Abeel, Thomas, de, Y. V., & Yvan. (1970, January 1). Java-ML: A Machine Learning Library. Retrieved from http://www.jmlr.org/papers/v10/abeel09a.html.

2. Applied Information Inc. "What You Need to Know About the Smart Stop Lights Act of 2019." Applied Information Transportation Technology, appinfoinc.com/smart-stop-lights-act-of-2019/.

3. Austin, P. L. (2019, January 21). Want to Fix Traffic? Try Smarter Signal Lights. Retrieved October 21, 2019, from https://time.com/5502192/smart-traffic-lights-ai/.

4. Bazzan, A. L. C. (2005). A Distributed Approach for Coordination of Traffic Signal Agents. *Autonomous Agents and Multi-Agent Systems*, *10*(1), 131–164. doi: 10.1007/s10458-004-6975-9

5. Boston Transportation Department. (2018, July). Traffic Signal Operations Design Guidelines. Retrieved from https://www.boston.gov/sites/default/files/document-file-07-2018/btd_traffic_signal_operations_design_guidelines_july_2018.pdf.

6. Cambridge Traffic, Parking and Transportation Department. (2006). Traffic Control Signal Policy.

7. Dresner, K., & Stone, P. (2005). Multiagent traffic management. *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS 05*. doi: 10.1145/1082473.1082545

8. Gregoire, P.-L., Desjardins, C., Laumonier, J., & Chaib-Draa, B. (2007). Urban Traffic Control

Based on Learning Agents. *2007 IEEE Intelligent Transportation Systems Conference*. doi: 10.1109/itsc.2007.4357719

9. Mallawaarachchi, Vijini. "Introduction to Genetic Algorithms - Including Example Code." Introduction to Genetic Algorithms, Towards Data Science, 20 Nov. 2019, towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

10. Moor, G. (2019). Traffic Signal Design Terminology. Retrieved from https://www.traffic-signal-design.com/terminology_main.htm.

11. Tonguz, O., Viriyasitavat, W., & Bai, F. (2009). Modeling urban traffic: A cellular automata approach. *IEEE Communications Magazine*, *47*(5), 142–150. doi: 10.1109/mcom.2009.4939290

12. "Traffic Signals." *WSDOT*, 8 Oct. 2019, www.wsdot.wa.gov/Operations/Traffic/signals.htm.

13. Witten, I.H., Frank, E., Trigg, L., Hall, M., Holmes, G. & Cunningham, S.J. (1999). Weka: Practical machine learning tools and techniques with Java implementations. (Working paper 99/11). Hamilton, New Zealand: University of Waikato, Department of Computer Science.

Appendices

Project Notes

Project Title: Coordination of Traffic Signals in Rural Areas with Machine learning.

Name: Shane Ferrante

Contents:

Article #12 Notes: Traffic Signal Design Terminology

Article #13 Notes: Introduction to Genetic Algorithms

Article #14 Notes: What You Need to Know About The Smart Stop Lights Act of 2019

Knowledge Gaps:

This list provides a brief overview of the major knowledge gaps for this project, how they were resolved and where to find the information.

Knowledge Gap	Resolved By	Information is located	Date resolved
How to program in Python	**Udacity Online course**	**Udacity course**	**October**
How does Machine Learning work?	**Learning about the fundamentals of a genetic algorithm**	**Code/Project Notes**	**December**
How do Modern Traffic Lights work in Rural areas?	**Many Journal articles and websites.**	**Project Notes, Lit Review, STEM Thesis**	**No specific date**

Literature Search Parameters:

These searches were performed between (Start Date of reading) and XX/XX/2019.

List of keywords and databases used during this project.

Database/search engine	Keywords	Summary of search

Google Scholar	Traffic Management	With this broad search, I was able to find two good articles about Traffic Management and Modeling (Articles 1 and 2)
Google Scholar	Coordination of Traffic Lights	This search yielded many useful results including articles 3 and 6.
Google Scholar	Machine Learning in Java	This search showed many machine learning libraries in Java such as articles 4 and 5.

Article #1 Notes: Multiagent Traffic Management

Source Title	Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism
Source Author	K. Dresner, P. Stone
Source citation	Dresner, K., & Stone, P. (2005). Multiagent traffic management. *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS 05*. doi: 10.1145/1082473.1082545
Original URL	https://dl.acm.org/citation.cfm?id=1018799
Source type	Journal
Keywords	Traffic Management
Summary of key points	This paper talks about a different intersection control mechanism for large, busy cities that performs better than the standard traffic light.
Important Figures	1|C|viCt(i)-t0(i)
Reason for interest	This article provides insight into different criteria that I should take into account when creating my city model.
Notes	• Assumption of no collisions, cars cannot turn

	• Considers both average wait time AND worst-case scenario • In the figure shown, the average delay of an intersection is the sum of all delays of a car getting from point A to point B through the intersection divided by the number of cars in the intersection. • C - set of cars going through an intersection • vi - vehicle number i • t(i) - the actual time from point A to point B • t0(i) - the time it would take had there not been an intersection at all.
Follow up Questions	Why did the cars not turn in the model? How would cars having the option to turn change the results?

Article #2 Notes: Modeling Urban Traffic

Source Title	Modeling Urban Traffic: A Cellular Automata Approach
Source Author	O. Tonguz, W. Viriyasitavat, F. Bai
Source citation	Tonguz, O., Viriyasitavat, W., & Bai, F. (2009). Modeling urban traffic: A cellular automata approach. *IEEE Communications Magazine*, *47*(5), 142–150. doi: 10.1109/mcom.2009.4939290
Original URL	https://ieeexplore.ieee.org/abstract/document/4939290
Source type	Journal
Keywords	Traffic Management
Summary of key points	This article aims to demonstrate many different techniques for controlling traffic lights, and it shows that there is a significant difference in traffic dynamics between the different systems in urban areas.
Important Figures	Not Applicable

Reason for interest	The article provides a general overview of different strategies to improve traffic signal efficiency.
Notes	• Different traffic densities and situations tested for different times of day. • Tested cycle duration, green split, and coordination of traffic lights • Found that each of these three aspects play a major role in the efficiency of a traffic light. • More analysis in my STW Lit Review of this article
Follow up Questions	

Article #3 Notes: A market-inspired approach to reservation-based urban road traffic management

Source Title	A market-inspired approach to reservation-based urban road traffic management
Source Author	M. Vasirani, S. Ossowski
Source citation	Vasirani, M., and S. Ossowski. "A Market-Inspired Approach for Intersection Management in Urban Road Traffic Networks." *Journal of Artificial Intelligence Research*, vol. 43, 2012, pp. 621–659., doi:10.1613/jair.3560.
Original URL	http://www.aamas-conference.org/Proceedings/aamas09/pdf/01_Full%20Papers/10_54_FP_0182.pdf
Source type	Journal
Keywords	Urban Road Traffic Management
Summary of key points	Most current traffic models use flows to describe traffic, but this one uses specific cars that take up physical space in the model in a futuristic way. The article also gives an economic side to the story instead of a purely traffic based side.

Important Figures	Not Applicable
Reason for interest	Modeling of urban traffic is closely related to rural traffic which is likely what my project will be about
Notes	• Intersections communicate with each other • Drivers communicate with the light to "reserve physical space" (Vasirani 618) for their pass through the intersection • Automated traffic light managers coordinate and calculate when the driver will enter the light, and they adjust the light accordingly. • Machine Learning is used to optimize these adjustments. • From an economic perspective, this is greatly beneficial to drivers and also globally.
Follow up Questions	

Article #4 Notes: Java-ML: A Machine Learning Library

Source Title	Java-ML: A Machine Learning Library
Source Author	Thomas Abeel, Yves Van de Peer, Yvan Saeys
Source citation	Abeel, Thomas, de, Y. V., & Yvan. (1970, January 1). Java-ML: A Machine Learning Library. Retrieved from http://www.jmlr.org/papers/v10/abeel09a.html.
Original URL	http://www.jmlr.org/papers/v10/abeel09a.html
Source type	Website/Package for machine learning implementation
Keywords	Machine Learning Java
Summary of key points	This is a collection of algorithms for machine learning which is widely available, and easy to implement.

Important Figures	Not Applicable
Reason for interest	Java machine learning is going to be a key feature in my project.
Notes	http://java-ml.sourceforge.net/ Location of the download and more information
Follow up Questions	

Article #5 Notes: Weka

Source Title	Weka: Practical Machine learning tools and techniques with Java implementations
Source Author	Ian Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, S. Cunningham
Source citation	Witten, I.H., Frank, E., Trigg, L., Hall, M., Holmes, G. & Cunningham, S.J. (1999). Weka: Practical machine learning tools and techniques with Java implementations. (Working paper 99/11). Hamilton, New Zealand: University of Waikato, Department of Computer Science.
Original URL	https://researchcommons.waikato.ac.nz/handle/10289/1040
Source type	Website/Package for machine learning implementation
Keywords	Machine Learning Java
Summary of key points	This source is a collection of Java classes and libraries for machine learning. There is clear documentation of each of the algorithms and their uses, and it is usable regardless of the computer platform.
Important Figures	Not Applicable

Reason for interest	I need some sort of java machine learning library for my project unless I want to create machine learning from scratch.
Notes	Works similarly to Java-ML Very old and possibly outdated
Follow up Questions	

Article #6 Notes: A Distributed Approach for Coordination of Traffic Signal Agents

Source Title	A Distributed Approach for Coordination of Traffic Signal Agents
Source Author	Ana L. C. Bazzan
Source citation	Bazzan, A. L. C. (2005). A Distributed Approach for Coordination of Traffic Signal Agents. *Autonomous Agents and Multi-Agent Systems*, *10*(1), 131–164. doi: 10.1007/s10458-004-6975-9
Original URL	https://link.springer.com/article/10.1023/B:AGNT.0000049887.90232.cd
Source type	Journal
Keywords	Traffic Signal Coordination
Summary of key points	Uses evolutionary game theory to decide what the value of a specific algorithm may be. Each signal is working to reach individual goals while also taking a global goal into account.
Important Figures	Not Applicable
Reason for interest	An evolutionary genetic algorithm is possibly a way of solving my problem which I may use.

Notes	• Each signal it attempting to improve its own efficiency, while also attempting to reach global goal
Follow up Questions	How would the model change if the traffic lights disregarded individual goals and primarily focused on global goals?

Article #7 Notes: Urban Traffic Control Based on Learning Agents

Source Title	Urban Traffic Control Based on Learning Agents
Source Author	P. Gregoire, C. Desjardins, J. Laumonier, B. Chaib-draa
Source citation	Gregoire, P.-L., Desjardins, C., Laumonier, J., & Chaib-Draa, B. (2007). Urban Traffic Control Based on Learning Agents. *2007 IEEE Intelligent Transportation Systems Conference*. doi: 10.1109/itsc.2007.4357719
Original URL	http://www2.ift.ulaval.ca/~chaib/publications/ITSC07_0173_FI.pdf
Source type	Journal Article
Keywords	Optimization, Machine Learning, Reward
Summary of key points	It provides a dynamic traffic control system that can deal with different traffic densities and situations more optimally than pre-timed lights.
Important Figures	Not Applicable
Reason for interest	This article provides insight into how traffic lights currently work, as well as various techniques to use machine learning to train the lights.
Notes	Breaks down traffic into north and south versus east and west (Strictly Rectangular). Reinforcement Learning

Follow up Questions	

Article #8 Notes: Cambridge Signal Policy

Source Title	Traffic Control Signal Policy
Source Author	Cambridge Traffic, Parking and Transportation Department
Source citation	CITY OF CAMBRIDGE Traffic, Parking and Transportation Department. (2006). Traffic Control Signal Policy.
Original URL	https://www.cambridgema.gov/~/media/Files/Traffic/official-Signal-Policy.pdf?la=en
Source type	Official City Document
Keywords	Traffic Signal Coordination
Summary of key points	This document focused on many different things, but what was most fruitful was the section on how traffic lights are coordinated. Specifically point number six is about how intersections along arterial corridors are to be coordinated.
Important Figures	"6. Intersections along an arterial corridor will be coordinated to reduce delays and manage traffic volumes effectively. This will encourage use of arterial streets and discourage cutting through residential neighborhoods."
Reason for interest	This is an official document describing how traffic signals currently work.
Notes	In point number 6, it is clear that the coordination of signals is partially to reduce delays, but there are also two more concerns which are unique to urban areas. These are management of traffic volumes, and deterrence of cutting through residential neighborhoods. In urban areas, managing traffic volumes is very important because you do not want backup all the way to the next intersection. Also, large amounts of traffic from urban areas going through residential areas is not good for the people living in

	those areas. Both of these problems are lessened greatly by low traffic densities meaning that the only thing which the signals need to worry about is delays.
Follow up Questions	

Article #9 Notes: Boston Traffic Signal Design Guidelines

Source Title	Traffic Signal Operations Design Guidelines
Source Author	Boston Transportation Department
Source citation	Boston Transportation Department. (2018, July). Traffic Signal Operations Design Guidelines. Retrieved from https://www.boston.gov/sites/default/files/document-file-07-2018/btd_traffic_signal_operations_design_guidelines_july_2018.pdf.
Original URL	https://www.boston.gov/sites/default/files/document-file-07-2018/btd_traffic_signal_operations_design_guidelines_july_2018.pdf
Source type	Official City Document
Keywords	Traffic Signal Coordination
Summary of key points	This article goes into extensive depth about every aspect of traffic lights, but coordination is not mentioned once. The guidelines instead focus on proper data collection of intersections so that the light can be pretimed optimally.
Important Figures	Not Applicable
Reason for interest	This article shows how current “dumb” traffic lights work, and gives specifics on how lights are pretimed.
Notes	Very specific details are taken into account as well as data from other lights in a 1000ft radius, but the data is in terms of overall stats and not current stats.

Follow up Questions	

Article #10 Notes: Time Article

Source Title	Want to Fix Road Congestion? Try Smarter Traffic Lights.
Source Author	Austin, P. L.
Source citation	Austin, P. L. (2019, January 21). Want to Fix Traffic? Try Smarter Signal Lights. Retrieved October 21, 2019, from https://time.com/5502192/smart-traffic-lights-ai/.
Original URL	https://time.com/5502192/smart-traffic-lights-ai/
Source type	Tech Article from TIME
Keywords	"Smart Traffic Lights" "Coordination"
Summary of key points	The main point of this article is to describe how smarter traffic lights would be a great way to reduce traffic. It talks about the current system which entails how lights are pre-timed with traffic data from the area, and are optimized, but are unaware of anything which is going on at any given moment.
Important Figures	Not Applicable
Reason for interest	This is a technological article for a wider audience about the effectiveness of coordination of traffic lights.
Notes	Only three percent of lights are considered smart. Smart is considered anything which is aware of its situation at a given time. For instance, lights with sensors are smart as well as lights which are coordinated with other lights. Mentioned a study where lights along one street were coordinated.
Follow up Questions	

Article #11 Notes: General Information about Traffic signals

Source Title	Traffic Signals and Signal Coordination (Timing)
Source Author	Washington State Department of Transportation
Source citation	"Traffic Signals." *WSDOT*, 8 Oct. 2019, www.wsdot.wa.gov/Operations/Traffic/signals.htm.
Original URL	https://www.wsdot.wa.gov/Operations/Traffic/signals.htm
Source type	Website
Keywords	Signal Coordination General Information
Summary of key points	Gives information about the costs and reasons for installing new traffic lights. The article goes into great detail about when and why new traffic lights are installed in a FAQ-style manner.
Important Figures	Not Applicable
Reason for interest	This article gives a lot of information on the current state of traffic lights in 2019 in Washington State
Notes	Traffic lights are only installed when other forms of traffic management are proven to not be effective. Generally, there is an increase in rear-end collisions, but a decrease in more dangerous angle collisions. Talks about traffic light coordination along one single roadway.
Follow up Questions	

Article #12 Notes: Traffic Signal Design Terminology

Source Title	Traffic Signal Design Terminology
Source Author	Sanderson Associates
Source citation	“Traffic Signal Design Terminology.” *Traffic Signal Design Terminology*, www.traffic-signal-design.com/terminology_main.htm.
Original URL	https://www.traffic-signal-design.com/terminology_main.htm
Source type	Website
Keywords	Traffic Signal Terminology
Summary of key points	This article gives a summary of all major and important traffic light terms such as greensplit, greenwave, duration, and other valueable terminology
Important Figures	Not Applicable
Reason for interest	It is necessary that I know about all of the major traffic light jargon if I am going to be able to make a sucessful project.
Notes	Greensplit- the distribution of green and red lights over a duration Duration- The total amount of time it takes to run through a full light cycle Greenwave- The ofset of traffic lights in an arterial corridor which moves a platoon along a road.
Follow up Questions	

Article #13 Notes: Introduction to Genetic Algorithms

Source Title	Introduction to Genetic Algorithms
Source Author	Mallawaarachchi Vijini

Source citation	Mallawaarachchi, Vijini. "Introduction to Genetic Algorithms - Including Example Code." *Introduction to Genetic Algorithms*, Towards Data Science, 20 Nov. 2019, towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.
Original URL	https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
Source type	Website
Keywords	Genetic Algorithm
Summary of key points	This article gives an outline of what a genetic algorithm is and how it is used as well as an example of a simple genetic algorithm.
Important Figures	Not Applicable
Reason for interest	Since I am using a genetic algorithm for my project, I need to have a basic understanding of how to use a genetic algorithm.
Notes	Things needed: • Population • Organisms • Fitness function • Method of creating new generation from the previous
Follow up Questions	

Article #14 Notes: Smart Stop Lights Act of 2019

Source Title	What You Need to Know About The Smart Stop Lights Act of 2019
Source Author	Applied Information Inc.

Source citation	Applied Information Inc. "What You Need to Know About the Smart Stop Lights Act of 2019." *Applied Information Transportation Technology*, appinfoinc.com/smart-stop-lights-act-of-2019/.
Original URL	https://appinfoinc.com/smart-stop-lights-act-of-2019/
Source type	Website/Article
Keywords	Smart Traffic Signals
Summary of key points	This article talks about the Smart Stop Lights act of 2019 which will provide funding to major cities to implement smarter traffic control algorithms.
Important Figures	Not Applicable
Reason for interest	This article is very relevant to my project because it is providing funding for smarter traffic lights in urban areas.
Notes	3% of traffic lights in the US are considered smart. This act (currently in the legislative process) could theoretically provide ASCT to more areas.
Follow up Questions	

Project Proposal

Phrase 1: In most semi-rural areas, traffic lights all act independently without taking into account the status of other traffic lights leading to a lack of coordination and an increase in the total wait times and travel times of cars in these areas.

Phrase 2: The aim of this project is to engineer and simulate a modular algorithm that controls traffic lights more effectively than the current method by coordinating them to reduce the average time that cars spend waiting at red lights.

Background:

In most semi-rural areas, traffic lights all act independently without taking into account the status of other traffic lights leading to a lack of coordination and an increase in the total wait times and travel times of cars in these areas. If these traffic lights were coordinated with each other than they would be able to reduce the time they are stopped at lights. Most studies on coordination of traffic lights focus on urban areas with constant traffic at every single light. This project will focus on developing a model for machine learning that simulates rural areas where the traffic density is low enough where there is not constant backup at each light, yet there is enough traffic to where a lot of valuable time will be saved by coordination of traffic lights. The reason for this is because there is currently very little work being done on coordination of traffic lights in areas with low traffic densities despite the fact that it could potentially yield fruitful results. Although this model could theoretically be applied to urban areas, I will be training my model on a simulated area of low traffic density, so it will be optimized to a rural setting.

Currently, only about three percent of all traffic lights in the United States are considered “smart.” And even the ones that are may not communicate with other lights around them. A lot of these lights either are pre-timed based on the time of day through estimated traffic data in those areas (Austin 2019). Some smarter lights have sensors that detect cars and can simply switch the light from red to green if there is a car waiting for no reason. These sensors work effectively when there is no traffic coming from the other direction, but the problem with these is that they will only be useful in areas with very low traffic densities or at night. This is because these sensors are completely useless unless there is no traffic coming from the other direction. On the other side of the spectrum, large urban areas often will have coordinated traffic lights, but their function is mainly to minimize backup between intersections rather than to keep traffic moving steadily. For instance, the traffic control signal policy in Cambridge MA states that intersections along a busy street will be coordinated in order to reduce delays to discourage cutting through residential neighborhoods. All of these systems work well in their environments, but there is little use of these techniques in areas with traffic which is high enough that there is almost always a car in every intersection, but low enough that traffic never gets backed up too far. With the current traffic system, people are potentially losing a lot of time waiting at traffic lights which can be avoided. According to Boston’s traffic signal operations design guidelines, there is no mandate to coordinate traffic lights. Because of this, most towns simply use pre-timed lights which waste time. One assumption which my project is making is that there is a way for lights to physically communicate with each other and send data back and forth. Since my simulated model assumes perfect communication between lights that is something which must be taken into account.

Since this project is software-based and simulated, there are a few other aspects of the project to take into account. The first aspect of the project is how to effectively model traffic in a city or a town. There have been many studies before which have dealt with how to model traffic for me to take inspiration from. For example, O. Tonguz and W. Viriyasitavat (2009) created a model of urban traffic using

cellular automata. This means that a city was split up into cells large enough for exactly one car to fit in each cell. Although I will not be using cellular automata in my project, their work on cycle duration and the distribution of green and red lights over a single cycle will be very helpful when creating my city model. Another aspect of my project is the implementation of green wave. Every traffic light has a set amount of time which they will stay green and a set amount of time that they will stay red for any given time of day. However, in some smart traffic lights, this amount of time can be adjusted based on the traffic conditions or based on the traffic lights around them. This adjustment is called green wave and will be implemented into my project. The question about what makes one traffic system better than another is also very important to this project. For this, I need a fitness function that will tell me how one algorithm does in relation to another. To do this I will primarily be using a formula from an article entitled Multiagent Traffic Management by K. Dresner (2005) which gives the average delay of a given traffic light for each car which goes through that light.

The last aspect of my project is the programming language and type of machine learning that I will be using. In terms of programming language, Python, MatLab, and Java are all great languages that can handle machine learning with various pros and cons. One Pro about Python is that it is very useful for machine learning with programs like TensorFlow. MatLab is also known for its machine learning capabilities, but the problem with MatLab and Python are the fact that I am not familiar with the languages, and learning to code will only add to the difficulty of the project. Java on the other hand is a language that I am familiar with, but it does not have the same machine learning capabilities of MatLab and Python. However, there are a few machine learning libraries for Java such as Weka (Witten 1999) and Java-ML (Abeel 1970) which, despite their age, work just as well as they did when they were created. For machine learning, many machine learning techniques are out there such as a Genetic Algorithm, Deep Learning, and Reinforcement Learning. Although deep learning and reinforcement learning are very useful tools for deep computational machine learning, this project will likely use a

genetic algorithm for two reasons. The first reason is that there is no need for a deep neural network for what I am attempting to accomplish. The second reason is that if I was to use Java then a genetic algorithm would be much easier because high level machine learning in Java does not work very well.

Project Definition:

The aim of this project is to engineer and simulate a modular algorithm that controls traffic lights more effectively than the current method by coordinating them to reduce the average time that cars spend waiting at red lights. To do this I will model a small town with a medium to low traffic density in which my algorithm will be tested. This model will be expandable to larger-scale towns and will be a simplified model of normal driving. Then, I will use machine learning through either Deep Learning, Reinforcement Learning, or a Genetic Algorithm to coordinate the traffic lights in the town by running simulations of the model. After the algorithm is optimized, it will be compared to the currently popular method of pre-timed lights to compare wait times of cars in each system to see how effective properly coordinated lights is on traffic. I predict that on average, cars will save time through coordinated traffic lights as opposed to uncoordinated traffic lights.

Experimental Design/Research Plan Goals:

My project will be divided into three phases. The first phase of the project will be modeling. This includes deciding on a programming language and a type of machine learning. Then, I will model a small network of roads and intersections, and create a method for adding cars to the network and telling them to drive from one point to another. The second phase will be implementing a general modifiable algorithm to coordinate the lights. Each light will have a specified baseline duration which it will be

green and a specified duration it will be red. It will also have a factor which will change slightly modify the duration of a green or a red light based on the current status of other lights. Then in the third phase, I will use machine learning to train the algorithm to minimize the average time spent at a red light per car. I will then compare this to the current system of independent lights to see how coordinating lights affects average wait time.

Timeline:

Research - Until the end of September

Decide on a Programming language - By the end of September

Decide on a type of machine learning - By the end of October

Create a model of a city - By the end of November

Implement the current uncoordinated algorithm - By the end of December

Implement coordinated AI with machine learning - By the end of January

Code

```
public class Algorithm {

	private TrafficLight[][] lights;
	private double[][][] parameters;
	//0-3: UDLR Target
	//4-7: UDLR Factor

	public Algorithm(TrafficLight[][] lights) {
		this.lights = lights;
		parameters = new double[lights.length][lights[0].length][8];
		generateRandomParameters();
		giveParameters();
	}

	public Algorithm(TrafficLight[][] lights, double[][][] parameters) {
		this.lights = lights;
		this.parameters = parameters;
		giveParameters();
	}

	public void generateRandomParameters() {
		for (int i = 0; i < lights.length; i++) {
			for (int j = 0; j < lights[0].length; j++) {
				for (int k = 0; k < 4; k++) {
					parameters[i][j][k] =
Math.random()*Parameters.LIGHT_DEFAULT_TIME-Parameters.LIGHT_DEFAULT_TIME/2;
				}
				for (int k = 4; k < 8; k++) {
					parameters[i][j][k] = Math.random();
```

```
                    }
                }
            }
        }

        public void mutate(double mutationFactor) {
            for (int i = 0; i < parameters.length; i++) {
                for (int j = 0; j < parameters[0].length; j++) {
                    for (int k = 0; k < 4; k++) {
                        if (Math.random() < mutationFactor) {
                            parameters[i][j][k] =
Math.random()*Parameters.LIGHT_DEFAULT_TIME-Parameters.LIGHT_DEFAULT_TIME/2;
                        }
                    }
                    for (int k = 4; k < 8; k++) {
                        if (Math.random() < mutationFactor) {
                            parameters[i][j][k] = Math.random();
                        }
                    }
                }
            }
        }

        public double[][][] getParameters() {
            return parameters;
        }

        public void giveParameters() {
            for (int i = 0; i < lights.length; i++) {
                for (int j = 0; j < lights[0].length; j++) {
```

```
                                        lights[i][j].setParameters(parameters[i][j]);
                                }
                        }
                }

}
import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;

public class Car {

        //Fields
        private Road road;
        private boolean directionTop;
        private int ticksRemaining;
        private int routePosition;
        private City city;
        private ArrayList<Node> route = new ArrayList<Node>();
        private Node startNode, endNode;
        private boolean travelling;
        private int waitTime = 0;

        //Constructors
        public Car(City city, Node startNode, Node endNode) {
                this.city = city;
                this.startNode = startNode;
                this.endNode = endNode;
                route = city.getRoute(startNode, endNode);
                road = city.getConnection(route.get(0), route.get(1));
```

```
            directionTop = road.getTopNode().equals(route.get(1));
            ticksRemaining = Road.length;
            routePosition = 0;
            travelling = true;
        }

        //Methods
        private void printRoute() {
            for (Node node : route) {
                System.out.print(node.getIndex()+" ");
            }
            System.out.println();
        }

        public void tick() {
            if (ticksRemaining > 0) ticksRemaining--;
            else {
                if (route.get(routePosition+1)==endNode) {
                    travelling = false;
                }
                else {
                    if (route.get(routePosition+1).getLight()==null ||
route.get(routePosition+1).getLight().isGreen(road)) {
                        routePosition++;
                        ticksRemaining = Road.length;
                        road = city.getConnection(route.get(routePosition),
route.get(routePosition+1));
                        directionTop =
road.getTopNode().equals(route.get(routePosition+1));
                    }
```

```
                                else if (!route.get(routePosition+1).getLight().isGreen(road)) {
                                        waitTime++;
                                }
                        }
                }
        }

        public int getTrafficLights() {
                if (route.size() >= 2) {
                        return route.size()-2;
                }
                return 0;
        }

        public int getWaitTime() {
                return waitTime;
        }

        public boolean isTravelling() {
                return travelling;
        }

        //Draw Function
        public void draw(Graphics g) {
                g.setColor(Color.BLUE);
                int xPos, yPos;
                if (directionTop) {
                        xPos = (int)(road.getBottomNode().getX()*ticksRemaining/(double)Road.length
+
                                        road.getTopNode().getX()*(Road.length-
ticksRemaining)/(double)Road.length);
```

```
                        yPos = (int)(road.getBottomNode().getY()*ticksRemaining/(double)Road.length
+
                                        road.getTopNode().getY()*(Road.length-
ticksRemaining)/(double)Road.length);
                }
                else {
                        xPos = (int)(road.getTopNode().getX()*ticksRemaining/(double)Road.length +
                                        road.getBottomNode().getX()*(Road.length-
ticksRemaining)/(double)Road.length);
                        yPos = (int)(road.getTopNode().getY()*ticksRemaining/(double)Road.length +
                                        road.getBottomNode().getY()*(Road.length-
ticksRemaining)/(double)Road.length);
                }
                g.fillRect(xPos, yPos, Parameters.CAR_SIZE, Parameters.CAR_SIZE);
        }
}
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.Random;

public class City {

        //Fields
        private Layout layout;
        private ArrayList<Node> nodes;
        private Window window;
        private ArrayList<Car> cars = new ArrayList<Car>();
        private int totalTicks;
        private int totalWaitTime;
        private int totalTrafficLights;
        private Algorithm alg;
        private double[][][] parameters;
```

```
Random rand = new Random();

//Constructors
public City(Layout layout, int lights) {
        this.layout = layout;
        this.nodes = layout.getNodes();
        createLights(lights);
        TrafficLight[][] lightsArr = createLightsArr();
        alg = new Algorithm(lightsArr);
        totalTicks = 0;
        //start();
}

public City(Layout layout, int lights, double[][][] parameters) {
        this.layout = layout;
        this.nodes = layout.getNodes();
        createLights(lights);
        TrafficLight[][] lightsArr = createLightsArr();
        alg = new Algorithm(lightsArr, parameters);
        totalTicks = 0;
        //start();
}

//Methods
public void addWindow(Window window) {
        this.window = window;
}

public void mutate(double mutationFactor) {
        alg.mutate(mutationFactor);
```

```
	}

	public TrafficLight[][] createLightsArr() {
		TrafficLight[][] lightsArr = new TrafficLight[Parameters.ROWS][Parameters.COLUMNS];
		for (int i = 0; i < Parameters.ROWS; i++) {
			for (int j = 0; j < Parameters.COLUMNS; j++) {
				lightsArr[i][j] = nodes.get(i*Parameters.COLUMNS+j).getLight();
			}
		}
		return lightsArr;
	}

	public void createLights(int lights) {
		for (int i = 0; i < lights; i++) {
			nodes.get(i).addLight();
		}
	}

	public Road getConnection(Node node1, Node node2) {
		for (Road road : node1.getConnections()) {
			if ((road.getTopNode().equals(node1) &&
road.getBottomNode().equals(node2)) ||
				(road.getTopNode().equals(node2) &&
road.getBottomNode().equals(node1))) {
				return road;
			}
		}
		System.out.println("Error");
		return null;
	}
```

```
public ArrayList<Node> getRoute(Node startNode, Node endNode) {
        ArrayList<ArrayList<Node>> paths = new ArrayList<ArrayList<Node>>();
        ArrayList<Node> start = new ArrayList<Node>();
        start.add(startNode);
        int length = 0;
        paths.add(start);
        while(length < Parameters.MAX_LENGTH) {
                ArrayList<ArrayList<Node>> newPaths = new ArrayList<ArrayList<Node>>();
                for (ArrayList<Node> path : paths) {
                        Node lastNode = path.get(path.size()-1);
                        for (Road road : lastNode.getConnections()) {
                                ArrayList<Node> newPath = new ArrayList<Node>();
                                for (Node node : path) {
                                        newPath.add(node);
                                }
                                if (road.getTopNode().equals(lastNode)) {
                                        newPath.add(road.getBottomNode());
                                }
                                if (road.getBottomNode().equals(lastNode)) {
                                        newPath.add(road.getTopNode());
                                }
                                if (newPath.get(newPath.size()-1).equals(endNode)) {
                                        return newPath;
                                }
                                if (!path.contains(newPath.get(newPath.size()-1))) {
                                        newPaths.add(newPath);
                                }
                        }
                }
                paths = newPaths;
```

```
                length++;
        }
        System.out.println("Error");
        return null;
}

public void start() {
        while(totalTicks < Parameters.TICKSPERTRIAL) {
                tick();
                if (Parameters.MODE.equals("Display")) {
                        window.repaint();
                }
                try {
                        if (Parameters.SLEEP_TIME > 0) Thread.sleep(Parameters.SLEEP_TIME);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
}

public double getFitness() {
        return totalWaitTime/(double)totalTrafficLights;
}

public void tick() {
        totalTicks++;
        for (Node node : nodes) {
                node.tick();
        }
```

```
		for (int i = 0; i < cars.size(); i++) {
			cars.get(i).tick();
			if (!cars.get(i).isTravelling()) {
				totalTrafficLights += cars.get(i).getTrafficLights();
				totalWaitTime += cars.get(i).getWaitTime();
				cars.remove(i);
				i--;
			}

		}
		if (totalTicks % Parameters.TIME_PER_CAR == 0) {
			if (cars.size() < Parameters.MAX_CARS) createCar();
		}
	}

	public void printInfo() {
		System.out.println("Total Ticks " + totalTicks);
		System.out.println("Total Wait Time: " + totalWaitTime);
		System.out.println("Total Traffic Lights: " + totalTrafficLights);
		System.out.println("Avg Wait Time/Light: " + totalWaitTime/(double)totalTrafficLights);
		System.out.println();
	}

	public void createCar() {
		int node1 = rand.nextInt(nodes.size());
		int node2 = rand.nextInt(nodes.size());
		if (node1 == node2) {
			createCar();
		}
		else {
```

```
                        Car newCar = new Car(this, nodes.get(node1), nodes.get(node2));
                        cars.add(newCar);
                }
        }

        public double [][][] getParameters() {
                return alg.getParameters();
        }

        //Draw Function
        public void draw(Graphics g) {
                layout.draw(g);
                for (Car car : cars) {
                        car.draw(g);
                }
        }
}
import java.util.ArrayList;
import java.util.Random;

public class GeneticAlgorithm {

        private City[] population;
        private int popSize;
        private Layout layout;
        private int lights;
        private int generationCount;
        private double mutationFactor = 0.05;
        private Random random;
        private ArrayList<Double> avgFits;
```

```
	private Window window;

	public GeneticAlgorithm(int popSize, Layout layout, int lights, Window window) {
		this.window = window;
		this.popSize = popSize;
		this.layout = layout;
		this.lights = lights;
		this.random = new Random();
		generationCount = 0;
		avgFits = new ArrayList<Double>();
		createPopulation();
		int count = 5;
		while (count > 0) {
			doGeneration();
			avgFits.add(getAverageFit());
			if (avgFits.size() > 15 && avgFits.get(avgFits.size()-1) > avgFits.get(avgFits.size()-
15)) count--;
			if (count == 0) {
				doGeneration();
				for (int i = 0; i < popSize; i++) {
					System.out.print(population[i].getFitness() + " ");
				}
				System.out.println();
				doGeneration();
				for (int i = 0; i < popSize; i++) {
					System.out.print(population[i].getFitness() + " ");
				}
				System.out.println();
				doGeneration();
				for (int i = 0; i < popSize; i++) {
```

```
                                        System.out.print(population[i].getFitness() + " ");
                                }
                                System.out.println();
                        }
                }
        }

        public void doGeneration() {
                generationCount++;
                City[] fittest = sortPopulation();
                City[] newPopulation = generateNewPopulation(fittest);
                mutatePopulation(newPopulation);
                population = newPopulation;
                printInfo();
        }

        public void mutatePopulation(City[] newPopulation) {
                for (int i = 0; i < newPopulation.length; i++) {
                        newPopulation[i].mutate(mutationFactor);
                }
        }

        public City cross(City parent1, City parent2) {
                double[][][] params1 = copyArr(parent1.getParameters());
                double[][][] params2 = copyArr(parent2.getParameters());
                double[][][] newParams = new
double[params1.length][params1[0].length][params1[0][0].length];
                for (int i = 0; i < params1.length; i++) {
                        for (int j = 0; j < params1[0].length; j++) {
                                for (int k = 0; k < params1[0][0].length; k++) {
```

```
                                        newParams[i][j][k] = (params1[i][j][k] + params2[i][j][k])/2;
                                }
                        }
                }
                return new City(layout, lights, newParams);
        }

        public double[][][] copyArr(double[][][] arr) {
                double[][][] copy = new double[arr.length][arr[0].length][arr[0][0].length];
                for (int i = 0; i < arr.length; i++) {
                        for (int j = 0; j < arr[0].length; j++) {
                                for (int k = 0; k < arr[0][0].length; k++) {
                                        copy[i][j][k] = arr[i][j][k];
                                }
                        }
                }
                return copy;
        }

        public City[] generateNewPopulation(City[] populationF) {
                City[] newPopulation = new City[popSize];
                for (int i = 0; i < popSize/4; i++) {
                        newPopulation[i] = new City(layout, lights, populationF[i].getParameters());
                        //newPopulation[i].addWindow(window);
                }
                for (int i = popSize/4; i < popSize; i++) {
                        City parent1 = populationF[random.nextInt(popSize/2)];
                        City parent2 = populationF[random.nextInt(popSize/2)];
                        newPopulation[i] = cross(parent1, parent2);
                }
```

```
        return newPopulation;
}

public City[] sortPopulation() {
        ArrayList<City> sorted = new ArrayList<City>();
        ArrayList<Double> fits = new ArrayList<Double>();
        for (int i = 0; i < popSize; i++) {
                double fit = population[i].getFitness();
                int counter = 0;
                while(counter < sorted.size() && sorted.get(counter).getFitness() < fit) {
                        counter++;
                }
                if (counter < sorted.size()) {
                        sorted.add(counter, population[i]);
                        fits.add(counter, fit);
                }
                else {
                        sorted.add(population[i]);
                        fits.add(fit);
                }
        }
        //System.out.println(fits);
        City[] fittest = new City[popSize];
        for (int i = 0; i < popSize; i++) {
                fittest[i] = sorted.get(i);
        }
        return fittest;
}

public void createPopulation() {
```

```
        population = new City[popSize];
        for (int i = 0; i < popSize; i++) {
                population[i] = new City(layout, lights);
                //population[i].addWindow(window);
                population[i] = new City(layout, lights, population[i].getParameters());
                //population[i].addWindow(window);
                population[i].start();
        }
        printInfo();
}

public void printInfo() {
        System.out.println("--------------------");
        System.out.println("Generation: " + generationCount);
        System.out.println("Average Fit: " + getAverageFit());
        System.out.println("Best Fit: " + getBestFit());
        System.out.println("Worst Fit: " + getWorstFit());
        System.out.println("STDEV: " + getStdev());
        if (generationCount == 0) {
                for (int i = 0; i < popSize; i++) {
                        System.out.print(population[i].getFitness() + " ");
                }
                System.out.println();
        }
}

public double getAverageFit() {
        double sum = 0;
        for (City city : population) {
                sum += city.getFitness();
```

```
        }
        return sum/popSize;
}

public double getStdev() {
        double average = getAverageFit();
        double sum = 0;
        for (City city : population) {
                sum += Math.pow(city.getFitness()-average, 2);
        }
        return Math.sqrt(sum/population.length);
}

public double getBestFit() {
        double best = population[0].getFitness();
        for (City city : population) {
                if (city.getFitness() < best) best = city.getFitness();
        }
        return best;
}

public double getWorstFit() {
        double best = population[0].getFitness();
        for (City city : population) {
                if (city.getFitness() > best) best = city.getFitness();
        }
        return best;
}

public int getPopSize() {
```

```
		return popSize;
	}

	public City[] getPopulation() {
		return population;
	}

}
import java.awt.Graphics;
import java.util.ArrayList;

public class Layout {

	//Fields
	ArrayList<Node> nodes = new ArrayList<Node>();

	//Constructors
	public Layout(int[][] nodePoints, int[][] connections) {
		createNodes(nodePoints);
		createConnections(connections);
	}

	//Methods
	public void createNodes(int[][] nodePoints) {
		for (int i = 0; i < nodePoints.length; i++) {
			nodes.add(new Node(nodePoints[i][0], nodePoints[i][1], i, this));
		}
	}

	public void createConnections(int[][] connections) {
```

```
                for (int i = 0; i < connections.length; i++) {
                        Road road = new Road(nodes.get(connections[i][0]),
nodes.get(connections[i][1]), connections[i][2]);
                        nodes.get(connections[i][0]).addConnection(road);
                        nodes.get(connections[i][1]).addConnection(road);
                }
        }

        public void printNodeInfo() {
                for (Node node : nodes) {
                        System.out.println("Node " + node.getIndex() + ": " +
node.getConnections().size());
                }
        }

        //Getters and Setters
        public Node getNode(int num) {
                return nodes.get(num);
        }

        public ArrayList<Node> getNodes() {
                return nodes;
        }

        //Draw Function
        public void draw(Graphics g) {
                for (Node node : nodes) {
                        node.draw(g);
                }
        }
}
```

```
import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;

public class Node {

	//Fields
	private int x, y, index;
	private ArrayList<Road> connections = new ArrayList<Road>();
	private Layout layout;
	private TrafficLight light;

	//Constructors
	public Node(int x, int y, int index, Layout layout) {
		this.x = x;
		this.y = y;
		this.index = index;
		this.layout = layout;
	}

	//Methods
	public void addConnection(int num, int direction) {
		connections.add(new Road(this, layout.getNode(num), direction));
	}

	public void addConnection(Road road) {
		connections.add(road);
	}

	public void addLight() {
```

```
            light = new TrafficLight(this, Parameters.LIGHT_DEFAULT_TIME);
        }

        public void tick() {
            if (light != null) {
                light.tick();
            }
        }

        //Getters and Setters
        public int getX() {
            return x;
        }

        public int getY() {
            return y;
        }

        public int getIndex() {
            return index;
        }

        public Node getUp() {
            for (Road road : connections) {
                if (road.getBottomNode().equals(this) && road.getDirection() == Road.N) return
road.getTopNode();
            }
            return null;
        }
```

```
	public Node getDown() {
		for (Road road : connections) {
			if (road.getTopNode().equals(this) && road.getDirection() == Road.N) return road.getBottomNode();
		}
		return null;
	}

	public Node getRight() {
		for (Road road : connections) {
			if (road.getBottomNode().equals(this) && road.getDirection() == Road.E) return road.getTopNode();
		}
		return null;
	}

	public Node getLeft() {
		for (Road road : connections) {
			if (road.getTopNode().equals(this) && road.getDirection() == Road.E) return road.getBottomNode();
		}
		return null;
	}

	public ArrayList<Road> getConnections() {
		return connections;
	}

	public TrafficLight getLight() {
		return light;
	}
```

```
	public boolean hasLight() {
		return !(light == null);
	}

	//Draw Function
	public void draw(Graphics g) {
		int size = Parameters.lightSize;
		g.setColor(Color.BLACK);
		g.fillRect(x-10*size/20, y-10*size/20, 20*size/20, 20*size/20);
		g.drawString("" + index, x-20*size/20, y-10*size/20);
		for (Road road : connections) {
			if (road.getTopNode().equals(this)) {
				g.drawLine(road.getTopNode().getX(), road.getTopNode().getY(),
road.getBottomNode().getX(), road.getBottomNode().getY());
			}
		}
		if (light != null) {
			light.draw(g);
		}
	}
}

public class Parameters {

	//Display MODE
	public static final String MODE = "Display";
	public static final int HEIGHT = 950;
	public static final int WIDTH = 1900;
	public static final int ROWS = 2;
```

```
        public static final int COLUMNS =3;
        public static final int lightSize = 40;
        public static final int MAX_LENGTH = ROWS+COLUMNS+2;
        public static final int SLEEP_TIME = 400;
        public static final int TIME_PER_CAR = 1;
        public static final int GENS = 25;
        public static final int DENSITY = 2;
        public static final int MAX_CARS = ROWS*COLUMNS*DENSITY;
        public static final int CAR_SIZE = 5;
        public static final double LIGHT_DEFAULT_TIME = 30;
        public static final int CROSS_PROBABILITY = 10000;
        public static final double CROSS_TIME = 10;
        public static final int TICKSPERTRIAL = 1000000;

        //Trial MODE
//      public static final String MODE = "Trial";
//      public static final int HEIGHT = 800;
//      public static final int WIDTH = 1200;
//      public static final int ROWS = 3;
//      public static final int COLUMNS =3;
//      public static final int MAX_LENGTH = ROWS+COLUMNS+2;
//      public static final int SLEEP_TIME = 0;
//      public static final int TIME_PER_CAR = 1;
//      public static final int GENS = 25;
//      public static final int DENSITY = 6;
//      public static final int MAX_CARS = ROWS*COLUMNS*DENSITY;
//      public static final int CAR_SIZE = 3;
//      public static final double LIGHT_DEFAULT_TIME = 30;
//      public static final int CROSS_PROBABILITY = 10000;
//      public static final double CROSS_TIME = 10;
```

```
//	public static final int TICKSPERTRIAL = 100000;

}

public class Road {

	//Fields
	public static final int length = 10;
	private Node topNode;
	private Node bottomNode;
	private int directionTop; //Either 0, 1, 2, 3
	private int directionBottom;

	public static final int N = 0;
	public static final int E = 1;
	public static final int S = 2;
	public static final int W = 3;

	public Road(Node topNode, Node bottomNode, int direction) {
		this.topNode = topNode;
		this.bottomNode = bottomNode;
		this.directionTop = direction;
		this.directionBottom = (direction + 2)%4;
	}

	public Node getTopNode() {
		return topNode;
	}

	public Node getBottomNode() {
```

```
		return bottomNode;
	}

	public Node getOtherNode(Node node) {
		if (node.equals(topNode)) return bottomNode;
		else if (node.equals(bottomNode)) return topNode;
		System.out.println("Error");
		return null;
	}

	public int getDirection() {
		return directionTop%2;
	}
}
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;

public class ToCSV {

	public static void main(String[] args) {
		File input = new
File("C:\\Users\\shane\\OneDrive\\Desktop\\workspace\\Traffic\\src\\in.txt");
		PrintWriter writer = null;
		Scanner sc = null;
		try {
			sc = new Scanner(input);
		} catch (FileNotFoundException e) {
```

```
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                try {
                        writer = new
PrintWriter("C:\\Users\\shane\\OneDrive\\Desktop\\workspace\\Traffic\\src\\out.csv");
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                while(sc.hasNextLine()) {
                        String generation = sc.nextLine();
                        generation += "\n" + sc.nextLine();
                        while(sc.hasNextLine()) {
                                String next = sc.nextLine();
                                if (next.equals("----------------------------------------------------------------------------
----------------------------------------------------------")) {
                                        break;
                                }
                                generation += "\n" + next;
                        }
                        runTrial(generation, writer);
                }
                sc.close();
                //writer.write("testMain");
                writer.close();
        }

        public static void runTrial(String trial, PrintWriter writer) {
                //writer.println("testTrial");
                ArrayList<String> avg = new ArrayList<String>();
```

```
ArrayList<String> best = new ArrayList<String>();
ArrayList<String> worst = new ArrayList<String>();
ArrayList<String> stdev = new ArrayList<String>();
Scanner sc = new Scanner(trial);
String startLine = sc.nextLine();
writeStartingData(startLine, writer);
sc.nextLine();
sc.nextLine();
while (sc.hasNextLine()) {
	String generation = "";
	generation += sc.nextLine();
	while(sc.hasNextLine()) {
		String next = sc.nextLine();
		if (next.equals("----------")) {
			break;
		}
		generation += "\n" + next;
	}
	writeGeneration(generation, writer, avg, best, worst, stdev);
}
writer.print("Gen,");
for (int i = 0; i < avg.size(); i++) {
	writer.print(i+",");
}
writer.println();
writer.print("Avg,");
for (String val : avg) {
	writer.print(val + ",");
}
writer.println();
```

```
            writer.print("Best,");
            for (String val : best) {
                  writer.print(val + ",");
            }
            writer.println();
            writer.print("Worst,");
            for (String val : worst) {
                  writer.print(val + ",");
            }
            writer.println();
            writer.print("Stdev,");
            for (String val : stdev) {
                  writer.print(val + ",");
            }
            writer.println();
            writer.println();
            sc.close();
}

public static void writeStartingData(String info, PrintWriter writer) {
            ArrayList<String> cells = new ArrayList<String>();
            Scanner sc = new Scanner(info);
            cells.add("SIZE,");
            cells.add(sc.next());
            cells.add(sc.next());
            sc.next(); sc.next();
            cells.add(sc.next().toUpperCase() + ",");
            sc.next();
            cells.add(sc.next());
            cells.add("MUTATION_FACTOR,");
```

```
                sc.next();
                sc.next();
                sc.next();
                String mf = sc.next();
                cells.add(mf.substring(0, mf.length()-1)+",");
                String line = "";
                for (String cell : cells) {
                        line += cell;
                }
                writer.println(line);
                sc.close();
        }

        public static void writeGeneration(String generation, PrintWriter writer, ArrayList<String> avg,
ArrayList<String> best, ArrayList<String> worst, ArrayList<String> stdev) {
                //writer.write("testGen");
                Scanner sc = new Scanner(generation);
                sc.nextLine();
                String avgS = sc.nextLine();
                avg.add(avgS.substring(avgS.lastIndexOf(" ")+1));
                String bestS = sc.nextLine();
                best.add(bestS.substring(bestS.lastIndexOf(" ")+1));
                String worstS = sc.nextLine();
                worst.add(worstS.substring(worstS.lastIndexOf(" ")+1));
                String stdevS = sc.nextLine();
                stdev.add(stdevS.substring(stdevS.lastIndexOf(" ")+1));
                if (sc.hasNextLine()) {
                        writer.print("Gen " + (avg.size()-1) + " values,");
                        while(sc.hasNext()) {
                                writer.print(sc.next()+",");
```

```
                        }
                        writer.println();
                }
                sc.close();
        }

}
import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;

public class TrafficLight {

        private Node node;
        private double totalDuration;
        private int currentState; //0 for NS, 1 for WE, 2 for NONE
        private double timeLeft;
        private Random rand;
        private double[] parameters;

        public TrafficLight(Node node, double totalDuration) {
                this.node = node;
                this.totalDuration = totalDuration;
                currentState = 0;
                timeLeft = totalDuration/2;
                rand = new Random();
        }

        public void tick() {
                timeLeft--;
```

```
if (timeLeft <= 0) {
        if (rand.nextInt(Parameters.CROSS_PROBABILITY) == 0) {
                currentState = 2;
                timeLeft = Parameters.CROSS_TIME;
        }
        else {
                currentState = (currentState+1)%2;
                double currentDuration = totalDuration;
                Node currentNode = node.getUp();
                if (currentNode.hasLight()) {
                        TrafficLight currentLight = currentNode.getLight();
                        double l1 = getPos(), l2 = currentLight.getPos();
                        double x = l1-l2;
                        if (x < -totalDuration/2) x += totalDuration;
                        if (x > totalDuration/2) x -= totalDuration;
                        double y = parameters[0]-x;
                        if (y < -totalDuration/2) y += totalDuration;
                        if (y > totalDuration/2) y -= totalDuration;
                        currentDuration -= parameters[4]*y*0.25;
                }
                currentNode = node.getDown();
                if (currentNode.hasLight()) {
                        TrafficLight currentLight = currentNode.getLight();
                        double l1 = getPos(), l2 = currentLight.getPos();
                        double x = l1-l2;
                        if (x < -totalDuration/2) x += totalDuration;
                        if (x > totalDuration/2) x -= totalDuration;
                        double y = parameters[1]-x;
                        if (y < -totalDuration/2) y += totalDuration;
                        if (y > totalDuration/2) y -= totalDuration;
```

```
                    currentDuration -= parameters[5]*y*0.25;
                }
                currentNode = node.getLeft();
                if (currentNode.hasLight()) {
                    TrafficLight currentLight = currentNode.getLight();
                    double l1 = getPos(), l2 = currentLight.getPos();
                    double x = l1-l2;
                    if (x < -totalDuration/2) x += totalDuration;
                    if (x > totalDuration/2) x -= totalDuration;
                    double y = parameters[2]-x;
                    if (y < -totalDuration/2) y += totalDuration;
                    if (y > totalDuration/2) y -= totalDuration;
                    currentDuration -= parameters[6]*y*0.25;
                }
                currentNode = node.getUp();
                if (currentNode.hasLight()) {
                    TrafficLight currentLight = currentNode.getLight();
                    double l1 = getPos(), l2 = currentLight.getPos();
                    double x = l1-l2;
                    if (x < -totalDuration/2) x += totalDuration;
                    if (x > totalDuration/2) x -= totalDuration;
                    double y = parameters[3]-x;
                    if (y < -totalDuration/2) y += totalDuration;
                    if (y > totalDuration/2) y -= totalDuration;
                    currentDuration -= parameters[7]*y*0.25;
                }
                timeLeft = currentDuration/2;
            }
        }
    }
```

```
	public double getPos() {
		if (timeLeft > totalDuration/2) return currentState*totalDuration/2;
		return currentState*totalDuration/2 + totalDuration/2-timeLeft;
	}

	public void setParameters(double[] parameters) {
		this.parameters = parameters;
	}

	public boolean isGreen(Road road) {
		return currentState == road.getDirection();
	}

	public void draw(Graphics g) {
		int size = Parameters.lightSize;
		if (currentState == 0) {
			g.setColor(Color.GREEN);
			g.fillRect(node.getX()+3*size/20-10*size/20, node.getY()-10*size/20,
14*size/20, 3*size/20);
			g.fillRect(node.getX()+3*size/20-10*size/20, node.getY()+17*size/20-
10*size/20, 14*size/20, 3*size/20);
			g.setColor(Color.RED);
			g.fillRect(node.getX()-10*size/20, node.getY()+3*size/20-10*size/20, 3*size/20,
14*size/20);
			g.fillRect(node.getX()+17*size/20-10*size/20, node.getY()+3*size/20-
10*size/20, 3*size/20, 14*size/20);
		}
		else if (currentState == 1) {
			g.setColor(Color.RED);
			g.fillRect(node.getX()+3*size/20-10*size/20, node.getY()-10*size/20,
14*size/20, 3*size/20);
```

```
                        g.fillRect(node.getX()+3*size/20-10*size/20, node.getY()+17*size/20-
10*size/20, 14*size/20, 3*size/20);

                        g.setColor(Color.GREEN);

                        g.fillRect(node.getX()-10*size/20, node.getY()+3*size/20-10*size/20, 3*size/20,
14*size/20);

                        g.fillRect(node.getX()+17*size/20-10*size/20, node.getY()+3*size/20-
10*size/20, 3*size/20, 14*size/20);

                }

                else if(currentState == 2) {

                        g.setColor(Color.RED);

                        g.fillRect(node.getX()+3*size/20-10*size/20, node.getY()-10*size/20,
14*size/20, 3*size/20);

                        g.fillRect(node.getX()+3*size/20-10*size/20, node.getY()+17*size/20-
10*size/20, 14*size/20, 3*size/20);

                        g.setColor(Color.RED);

                        g.fillRect(node.getX()-10*size/20, node.getY()+3*size/20-10*size/20, 3*size/20,
14*size/20);

                        g.fillRect(node.getX()+17*size/20-10*size/20, node.getY()+3*size/20-
10*size/20, 3*size/20, 14*size/20);

                }

        }


}
import javax.swing.JFrame;

import javax.swing.plaf.metal.MetalToggleButtonUI;


public class TrafficRunner {


        public static void main(String[] args) {

                JFrame frame = new JFrame();

                Window window = new Window(Parameters.WIDTH, Parameters.HEIGHT);

                frame.setTitle("Traffic Coordination");

                frame.add(window);
```

```
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
        frame.setSize(Parameters.WIDTH + 23, Parameters.HEIGHT + 57);
        frame.setVisible(Parameters.MODE.equals("Display"));
        if (Parameters.MODE.equals("Display")) {
                Layout layout = createLayout(Parameters.ROWS, Parameters.COLUMNS);
                int lights = Parameters.ROWS*Parameters.COLUMNS;
                City city = new City(layout, lights);
                city.addWindow(window);
                window.setCity(city);
                city.start();
        }
        runTrial(window);
}

public static double average(double[] arr) {
        double sum = 0;
        for (double num : arr) {
                sum += num;
        }
        return sum/arr.length;
}

public static double stDev(double[] arr) {
        double average = average(arr);
        double sum = 0;
        for (double num : arr) {
                sum += Math.pow(num-average, 2);
        }
        return Math.sqrt(sum/arr.length);
}
```

```
	public static void runTrial(Window window) {
		Layout layout = createLayout(Parameters.ROWS, Parameters.COLUMNS);
		int lights = Parameters.ROWS*Parameters.COLUMNS;
//		City city = new City(layout, lights);
//		window.setCity(city);
//		city.start();
//		//city.printInfo();
//		return city;
		GeneticAlgorithm alg = new GeneticAlgorithm(100, layout, lights, window);
	}

	public static Layout createLayout(int rows, int columns) {
		int[][] nodePoints = createPoints(rows, columns);
		int[][] connections = createConnections(rows, columns);
		return new Layout(nodePoints, connections);
	}


	public static int[][] createPoints(int rows, int columns) {
		int numCities = rows*columns + 2*(rows+columns);
		int[][] nodePoints = new int[numCities][2];
		for (int i = 0; i < rows; i++) {
			for (int j = 0; j < columns; j++) {
				nodePoints[columns*i+j][0] = Parameters.WIDTH*(j+1)/(columns+1);
				nodePoints[columns*i+j][1] = Parameters.HEIGHT*(i+1)/(rows+1);
			}
		}
		for (int i = 0; i < columns; i++) {
			nodePoints[rows*columns+i][0] = Parameters.WIDTH*(i+1)/(columns+1);
```

```
                nodePoints[rows*columns+i][1] = Parameters.HEIGHT/(rows+1)/2;
            }
            for (int i = 0; i < columns; i++) {
                nodePoints[(rows+1)*columns+i][0] = Parameters.WIDTH*(i+1)/(columns+1);
                nodePoints[(rows+1)*columns+i][1] = Parameters.HEIGHT*rows/(rows+1) +
Parameters.HEIGHT/(rows+1)/2;
            }
            for (int i = 0; i < rows; i++) {
                nodePoints[(rows+2)*columns+i][0] = Parameters.WIDTH/(columns+1)/2;
                nodePoints[(rows+2)*columns+i][1] = Parameters.HEIGHT*(i+1)/(rows+1);
            }
            for (int i = 0; i < rows; i++) {
                nodePoints[(rows+2)*columns+rows+i][0] = Parameters.WIDTH/(columns+1)/2
+ Parameters.WIDTH*columns/(columns+1);
                nodePoints[(rows+2)*columns+rows+i][1] = Parameters.HEIGHT*(i+1)/(rows+1);
            }
            return nodePoints;
        }

        public static int[][] createConnections(int rows, int columns) {
            int numRoads = (rows+1)*columns+(columns+1)*rows;
            int[][] connections = new int[numRoads][3];
            for(int i = 0; i < rows; i++) {
                for (int j = 0; j < columns-1; j++) {
                    connections[i*(columns-1)+j][0] = i*columns+j;
                    connections[i*(columns-1)+j][1] = i*columns+j+1;
                    connections[i*(columns-1)+j][2] = Road.E;
                }
            }
            for(int i = 0; i < columns; i++) {
                for (int j = 0; j < rows-1; j++) {
```

```
                int completed = rows*(columns-1);
                connections[completed + i*(rows-1) + j][0] = j*columns+i;
                connections[completed + i*(rows-1) + j][1] = (j+1)*columns+i;
                connections[completed + i*(rows-1) + j][2] = Road.S;
        }
}
for (int i = 0; i < columns; i++) {
        int completed = rows*(columns-1) + columns*(rows-1);
        connections[completed + i][0] = rows*columns+i;
        connections[completed + i][1] = i;
        connections[completed + i][2] = Road.S;
}
for (int i = 0; i < columns; i++) {
        int completed = rows*(columns-1) + columns*(rows);
        connections[completed + i][0] = columns*(rows-1)+i;
        connections[completed + i][1] = columns*(rows+1)+i;
        connections[completed + i][2] = Road.S;
}
for (int i = 0; i < rows; i++) {
        int completed = rows*(columns-1) + columns*(rows+1);
        connections[completed + i][0] = columns*(rows+2)+i;
        connections[completed + i][1] = i*columns;
        connections[completed + i][2] = Road.E;
}
for (int i = 0; i < rows; i++) {
        int completed = rows*(columns) + columns*(rows+1);
        connections[completed + i][0] = (i+1)*columns-1;
        connections[completed + i][1] = (rows+2)*columns+rows+i;
        connections[completed + i][2] = Road.E;
}
```

```
                return connections;
        }

}
import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JPanel;

public class Window extends JPanel {

        private City city;
        private int width, height;

        public Window(int width, int height) {
                this.width = width;
                this.height = height;
                this.setSize(width, height);
        }

        public void setCity(City city) {
                city.addWindow(this);
                this.city = city;
        }

        public void paintComponent(Graphics g) {
                g.setColor(Color.WHITE);
                g.fillRect(0, 0, width, height);
                city.draw(g);
        }
```

```
}
```