A Novel Algorithm for Intelligent Traffic Light Control

Saeid AsgharzadehBonab

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
February 2017
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Mustafa Tümer
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Zeki Bayram
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Zeki Bayram

2. Assoc. Prof. Dr. Muhammed Salamah

3. Asst. Prof. Dr. Duygu Çelik Ertuğrul

ABSTRACT

Nowadays, increasing number of vehicles on the streets are causing congestion at the intersections and consequently resulting in disruptions in traffic flows. Overall responsibility for preventing congestion in intersections is on traffic light control systems. In order to provide an enhanced traffic light controlling system, we utilized VANET technology for accessing vehicles' travelling data and consequently traffic flow information. Our dynamic cycle traffic light management proposal provides an intelligent and real-time mechanism to choose the trafic lanes that will be allowed to pass through the intersection and adjust the green light duration in each cycle according to the traffic situation. The traffic density and short-term future traffic density computing in each road are two important steps of our proposed scheme. The broadcasted vehicles travelling data by using V2R communication helps us to compute traffic density values. Also, weather condition and important sites in the roads that make roads crowded are other key parameters in our scheme.

The proposed scheme for choosing traffic light cycle and adjusting traffic light is evaluated against the existing static and other intelligent systems using our developed simulation web application. We found that our introduced scheme has better performance than the other approaches by significantly reducing delay time and accordingly decreasing number of the stopped vehicles behind traffic lights.

Keywords: Traffic light controlling, Intersection congestion, Vehicular ad-hoc Network, Intelligent traffic light, Vehicle to infrastructure broadcasting.

ÖZ

Günümüzde, sayıları artan araçlar kavşaklarda tıkanıklığa yol açmakta ve bu nedenle tarafik akışında aksamalara neden olmaktadır. Kavşaklardaki tıkanıklığı önlemenin genel olarak sorumluluğunu trafik ışık kontrol sistemi almıştır. Gelişmiş bir trafik ışık kontrol sistemlerini elde etmek için, araçların seyehat bilgilerine ve dolayısı ile trafik akış bilgisine erişim şağlayan VANET teknolojisi kullandık. Dinamik döngü trafik ışık yönetimi önerimiz trafiğin o anki durumuna göre hangi şeritlerin kavşaktan geçmesine izin verileceğini kararlaştıran ve yeşil ışık süresini ayarlayan akıllı ve gerçek zamanlı bir mekanizma sağlamaktadır. Her bir yoldaki o andaki ve sonraki trafik yoğunluğu hesaplamaları önerilen planın iki önemli adımını oluşturmaktadır. Seyahat halindeki araçların V2R altyapısına gonderdikleri veriler trafik yoğunluğu değerlerini hesaplamamıza yardımcı oluyor. Buna ek olarak, hava koşulları ve yolların kalabalık olmasına neden olan önemli yerler planımızdaki diğer önemli faktörlerdir.

Trafik ışık döngüsünü belirlemek ve trafik ışıklarını ayarlamak için önerilen plan, geliştirilmiş olan simülasyon web uygulaması kullanılarak varolan statik ve diğer akıllı sistemlerle mukayese edilmiştir. Önerdiğimiz sistemin bekleme zamanını ve dolayısı ile trafik ışıkları arkasında bekleyen arabaların sayısını önemli ölçüde azaltması nedeniyle diğer yöntemlere göre daha iyi perfromans gösterdiği gözlemlenmiştir.

Anahtar Kelimeler: Trafik ışık kontrolü, Kavşak tıkanıklığı, Akıllı trafik ışıkları, Araçtan altyapıya yayın.

To My Family

ACKNOWLEDGMENT

Firstly, I wouldlike to thank Assoc. Prof. Dr. Zeki Bayram for his support and advice in the preparation of this study. Without his invaluable supervision and kindness, all my goals could have been stopped.

Prof. Dr Işık AYBAY, Chairman of the Department of Computer Engineering, Eastern Mediterranean University, helped me with various issues during the master studies and I am grateful to him.

I would like to thank my family who allowed and funded me to study in this university and supported me in the whole of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

VANET	Vehicular Ad-hoc Network
RSU	Road Side Unit
V2V	Vehicle to Vehicle
V2R	Vehicle to Road Side Unit
V2I	Vehicle to Infrastructure
ATLC	Algorithm for Traffic Light Control
ITLC	Intelligent Traffic Light Control
GPS	Global Positioning System
SA	Stop Area
FA	Free Area

Chapter 1

INTRODUCTION

Traffic congestion is an increasing problem for major metropolitan areas such as those in Turkey and other European countries. As the population of the world continues to grow, the problem of traffic congestion will unavoidably worsen. The cost of traffic congestion will increase because of the countless lost hours by the drivers in the roads due to congestion delays.

Traffic congestion has a negative effect on environment and health. In the environment, it increases fuel consumption and air pollution. In terms of health effects, it increases stress and mental ailment, which can effect on the quality of people's life. In addition, traffic congestion decelerates the transportation of goods, which increases price of the goods.

Traffic jam is a costly issue in the big cities that has brought about many serious problems, such as pollution, wasted time and psychological effects on people. Intersections are the most vital traffic bottlenecks in an urban area and solving their problems with a logical and integrated approach is a good step in organizing traffic jams on urban roads [1]. Due to the high volume of vehicles and multiple traffic intersections in metropolises, use of traditional methods for applying timing to the traffic lights is not responsive. This drawback is because of assigning fixed times to the different traffic lights in the whole day without paying attention to citizens' traffic

behaviourat different times of the day or without considering effects of seasonal changes in traffic volume on the roads. In addition, vehicles' volume behind the traffic light is not an important parameter in traditional methods, and it causes wasting fuel, increasing air pollution and wasting citizens times in the roads behind traffic lights [2]. In macro economic terms, it causes much loss in material resources and thousands of hours of citizens' time.

Fixed-time approach is the rule in urban areas for reasons of regularity and reducing unnecessary delay. In the fixed-time approach each traffic light has certain and fixed duration for green light and it can be changed manually or generated by central system.

It is clear that fixed time scheduling for traffic light control results in increasing traffic jam. Also, traffic congestion changes in different weather conditions, and fixed time scheduling cannot deal with that.

In the technological era and the current world; specifying a green time to each lane by traffic density in every moment and managing the urban transportation system are only possible through intelligent traffic control systems.

In the last few years, utilizing communication between vehicles has turned into a supportive approach to finding a solution to traffic congestions in intersections [3]. The innovation of VANETs gives another approach to controlling traffic in the metropolises. In this approach V2V and V2R connections are used to broadcast traffic status, predict future traffic congestion or compute density of each road [4].

In this thesis, we present "Algorithm for Traffic Light Control" (ATLC) for controlling and timing traffic lights in intersections. The purpose of the introduced method is to reduce delay time for each vehicle and accordingly to decrease traffic congestion in target intersections in the cities. We propose an algorithm for making the decision of "vehicles on which lane can pass the intersection" and a function for scheduling traffic lights by using data collected and broadcasted by RSUs. Our proposed algorithm aims to choose the best lane to have the green light. The scheduling function is used to calculate and set the best duration for each cycle of the green light. Some important sites in each lane such as stadiums, schools, bus stops or government offices are also used as input parameters to our algorithm. Additionally, we defined weather condition as an important factor in scheduling function. Other factors that used in our method are the number of stopped and moving cars behind a traffic light, traffic flow speed and distance between first and last stopped the car in each lane.

The rest of this thesis is organized as follows. Chapter 2 introduces existing algorithms for intelligent traffic light control. Chapter 3 presents our proposed new algorithm to select the best lanes to pass the intersection and the traffic light scheduling function. Chapter 4 presents a simulation environment for testing traffic light simulation algorithms. Simulation results are given in Chapter 5 where our proposal is compared with other schemes. Chapter 6 presents a discussion of the results for the investigated schemes. Finally, the conclusion is presented in Chapter 7.

Chapter 2

EXISTING ALGORITHMS FOR TRAFFIC LIGHT CONTROL

A large number of papers address control and timing of traffic lights in intersections, and several algorithms have been proposed using VANET technology to control traffic signals to enhance the traffic lights performance in intersections. In this chapter, we give brief information about many of the intelligent traffic light control algorithms that have been proposed. These algorithms use VANET to get traffic and vehicles' data as the key technology.

In [5], an Intelligent Traffic Lights Controlling system (ITLC) based on VANET is described. VANET helps in broadcasting the data related to the traffic and vehicles to the nearby vehicles. The traffic flow with largest density is scheduled first to set the phases of each traffic light cycle. The ready area is defined by the signalized road intersection to compute the allowable time for each cycle. This scheme introduces an Intelligent Traffic Light Controlling algorithm, which considers the real-time traffic characteristics of each traffic flow that intends to cross the road intersection of interest, whilst scheduling the time phases of each traffic light. The algorithm aims at increasing the traffic fluency by decreasing the waiting time of traveling vehicles at the signalized road intersections. Figure 2.1 illustrates algorithms proposed in this scheme. The main problem with this approach is focusing on one intersection.

Algorithm 1: Intelligent Traffic Light Scheduling Algorithm

Data: TL : traffic light; RA: ready area; d_i : the traffic density of the traffic flow i inside RA; t_i: the required time for all vehicles inside RA, at the traffic flow i, to cross the traffic intersection.

compute d_i and t_i of all traffic flows inside RA;
while d_i *of any of the traffic flows at* $TL > 0$ **do**
    let j the traffic flow with the maximum traffic density (d_j);
    let $i1$ and $i2$ the traffic flows that can cross the traffic intersection simultaneously with the traffic flow (j) ;
    if $d_{i1} > d_{i2}$ **then**
        P_{ji1} = schedule (j, $i1$);
        $d_{i1} = 0.0$; $t_{i1} = 0.0$;
    else
        P_{ji2} = schedule (j, $i2$);
        $d_{i2} = 0.0$; $t_{i2} = 0.0$;
    end
    $d_j = 0.0$; $t_j = 0.0$;
    Adjust the t_k of all other traffic flows inside the ready area;
end

Algorithm 2: Schedule Function

INPUT: traffic flows i and j
if $t_i > t_j$ **then**
    return t_i
else
    return t_j
end

Figure 2.1: Proposed Algorithms in ITLC

In [6], the introduced system uses both V2I and V2V communication schemes for traffic flows density estimation. Vehicles send data, including their direction, to each other, by exchanging a sequence of packets. Also, it contributes in controlling the traffic flow of next intersection by adding a field in vehicle's data.

In [10], an intelligent traffic light control approach is proposed with the goal of decreasing vehicles CO2 emissions, by VANET technology to get the traffic information of each surrounding traffic flow every moment. The traffic lights cycle and lights duration time is adjustined with the gathered traffic data and a recommendatory speed is provided to prevent unnecessary stops. The main issue of

this approach is isolating research on one intersection and not using vehicles information such as speed and position in scheduling traffic lights.

In [15], the proposed scheme implemented in-vehicle virtual traffic lights and uses vehicle-to-vehicle communications for migration of traffic lights. The introduced approach renders signalized control of intersections truly ubiquitous. First problem of this approach is huge costs for implementing and not addressing the important factors in the roads is second problem. In addition, the comparison with other approaches is not included in the simulation results.

In [16], the phase-based method uses improved detection data to find vehicle queues and optimizes traffic light duration in each time interval of 5 seconds in order to decrease vehicles' queue length behind a traffic light. In the proposed scheme, the methods of dynamic programming are used for optimization of the scheme.

In [17], the proposed scheme introduces a framework to learn the Traffic light cycles and timing information from low-sampling rate taxi GPS trajectories. The main problem in this scheme is that it does not use factors like weather.

In [18], the first approach is to identify traffic bottlenecks, by timing traffic lights in intersections. The second approach is to reduce the impact of traffic congestion by timing traffic lights at intersections in a regional network. In this scheme, the issue under scrutiny is inadequacy of factors like weather and not comparing with other proposed algorithms.

In [19], the introduced approach works with different statuses of vehicles on the road and in order to analyze the dependence of the traffic lights evaluation indexes, the special analysis on numerical statistics is introduced. Then, a real-coded genetic algorithm is proposed in order to solve the traffic light timing model. Finally, the proposed system calculates numerical results for algorithms. This scheme is concerned with the issue of isolated intersection analysis and effects of one intersection on another one are ignored.

In [20], the proposed approach introduces an intelligent control system for traffic signal applications, called Fuzzy Intelligent Traffic Signal control. A fuzzy logic based control is implemented. In order to analyze the effects of system, this approach tries to develop a computational framework to analyze the system using microscopic traffic simulation. This scheme is an attempt of using hardware with fuzzy logic based control. The main problem of this scheme is simulating in an isolated intersection and not using traffic data. In addition, the scheme is not compared with other schemes.

Chapter 3

PROPOSED NEW ALGORITHM

In this section, we introduce our approach (ATLC) that reasons and uses road data and vehicles traveller data to decide on which two lanes can pass the intersection and calculate the duration of green light in selected lanes. In the proposed scheme, we consider the typical four-way road intersections in the map with two lanes in each road, shown in Figure 3.1.

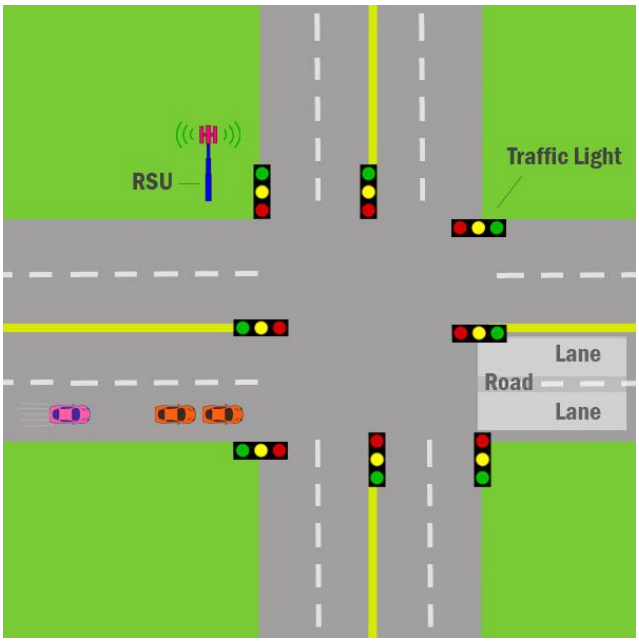

Figure 3.1: Intersection Structure with 4 Roads, 8 Input Lanes, 8 Output Lanes, 8 Traffic Lights, Vehicles and RSU

Table 3.1: Proposed scheme assumptions

Name	Description
Vehicle's speed	Vehicle's speed changing randomly by its position on the map
Vehicle's length	All the vehicles has same length

Assumptions are shown in Table 3.1. As shown in Figure 3.1, each intersection includes eight roads and one cross area, and each road includes two lanes. In the proposed scheme, we considered special coefficient for each road according to sites by the roads (e.g. stadiums, universities, markets, etc.). Road coefficient value is calculated by counting the number of important sites in each road and its value is in the range from 1 to 10. A typical intersection structure contains at most eight traffic flows in eight lanes such that half of the lanes are "flowing" into the intersection (input) and other half are "flowing" out (output). We assume that vehicles are moving on the map with random speed and they are choosing directions randomly according to the coefficient of each road. Also, we assume that vehicles have the same length and maximum speed. Each lane is virtually divided into two areas. "Stop Area" (SA) refers to the area behind a traffic light with stopped vehicles waiting to pass the intersection. "Free Area" FA refers to an area with vehicles moving on the lane after entering target lane. Figure 3.2 illustrates an example of SA and FA.

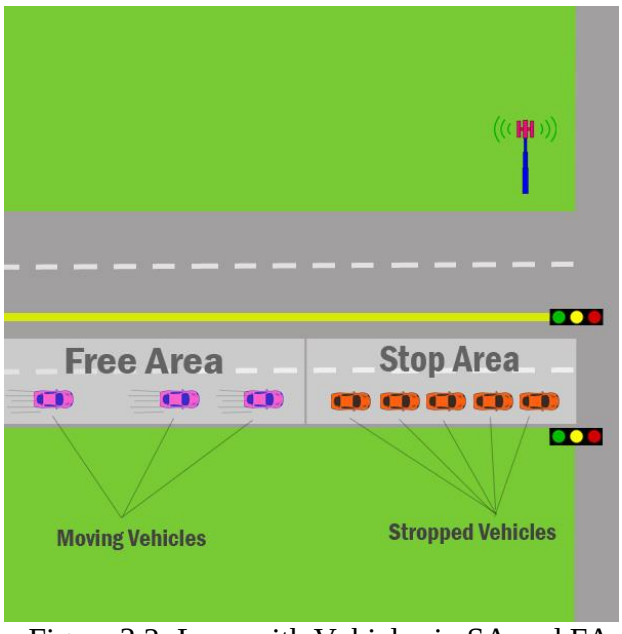

Figure 3.2: Lane with Vehicles in SA and FA

VANET technology helps us to gather important travelling data (i.e., position, destination, speed, etc.) [11]. Each road contains one RSU to gather travelling data periodically broadcasted by vehicles. Each RSU periodically sends vehicle and road data to our system to be used in our algorithm and scheduling function as in [7].

Table 3.2: Variables

Variable Name	Description
d_{si}	Density of the lane i in SA area
d_{fi}	Density of the lane i in FA area
s_i	Average speed of vehicles in lane i
E_i	Eligibility of lane i to have green light in its traffic light
m_i	Number of input lanes to lane i
w_i	Coefficient of lane i
C_i	Traffic flow speed in lane i
f	Weather condition coefficient
G_i	Green light duration in lane i
t	Startup delay time for first vehicle in the lane
R_i	Distance from traffic light to last vehicle in SA

Available variables in this scheme are shown in Table 3.2. The density of the lane i in SA (d_{si}), density of lane i in the FA (d_{fi}) and vehicles average speed in lane i (s_i) are calculated for each lane according to [8] using Equations 1, 2 and 3.

The eligibility (E) of each lane required to make a decision in selecting eligible lanes to pass the intersection is calculated by using Equation 1.

$$E_i = d_{si} + \alpha d_{fi} + \beta m_i + \gamma w_i \quad (1)$$

In this equation α, β and γ are constants between (0-1] and chosen randomly such that $\alpha > 2\beta$ and $\beta > \gamma$; m_i is the number of input lanes to the lane i; and, w_i is coefficient of the lane i that can be different from road to road. w_i is selected manually according to the available sites in the road.

In Equation 1, we are using d_{fi}, m_i and w_i in order to calculate short-term future congestion prediction. The variable d_{fi} helps us to predict a number of vehicles that will join stop area in the near future. The m_i and w_i can help us to find hidden vehicles that will enter the target lane in the near future [9].

The speed of the traffic flow in each lane is the average speed of all the vehicles but weather condition has effects on this value [21]. It is computed by Equation 2.

$$C_i = f \times s_i \quad (2)$$

where f is variable between (0-1] and refers to the weather condition. Minimum value of f is for the worst weather condition such as snowy weather. High value is for normal weather. s_i is the average speed of vehicles in lane i.

The green light duration G_i that allows the vehicles behind the traffic light in the lane i to cross the intersection is computed by using Equation 3.

$$G_i = t + E_i + (R_i/C_i) \quad (3)$$

where t is a constant value for startup delay of the first vehicle behind traffic light in the lane [14], R_i is the distance between the traffic light and farthest vehicle to traffic light in the SA, E_i is the eligibility in the lane i that is computed in Equation 1 and C_i is speed of traffic flow in the lane i that is computed by using Equation 2.

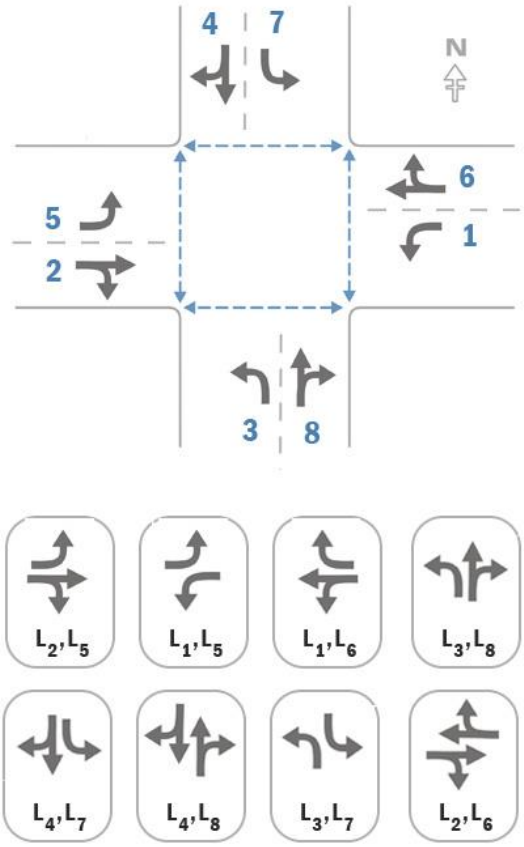

Figure 3.3: Possible Lanes Pairs to Make Traffic Lights States

A state is a pair of lanes that can legally pass the intersection. In our work, each intersection has at most eight lanes and eight states for crossing vehicles from intersection [12]. All the possible states illustrated in Figure 3.3. For our 8-lane intersection, depicted in Figure 3.3, the states are:

(L_1, L_5), (L_1, L_6), (L_2, L_5), (L_2, L_6), (L_3, L_7), (L_3, L_8), (L_4, L_7), (L_4, L_8)

To choose the best state, we use the data broadcasted by RSUs in order to make decision on states, as in [13]. The complete procedure of our proposed scheme is provided on a flow diagram in Figure 3.4 and includes the following steps:

Figure 3.4: Illustrating Steps of ATLC Scheme

Step 1: Compute base variables include density in SA, FA and average speed of vehicles in each lane.

Step 2: Compute E_i for each lane with data from Step 1.

Step 3: Choose best two lanes to have green light based on their eligibility value.

Step 4: Compute G for selected lane from Step 3 with maximum eligibility.

Step 5: Allow the two lanes selected in Step 3 to cross the intersection for the duration computed in Step 4.

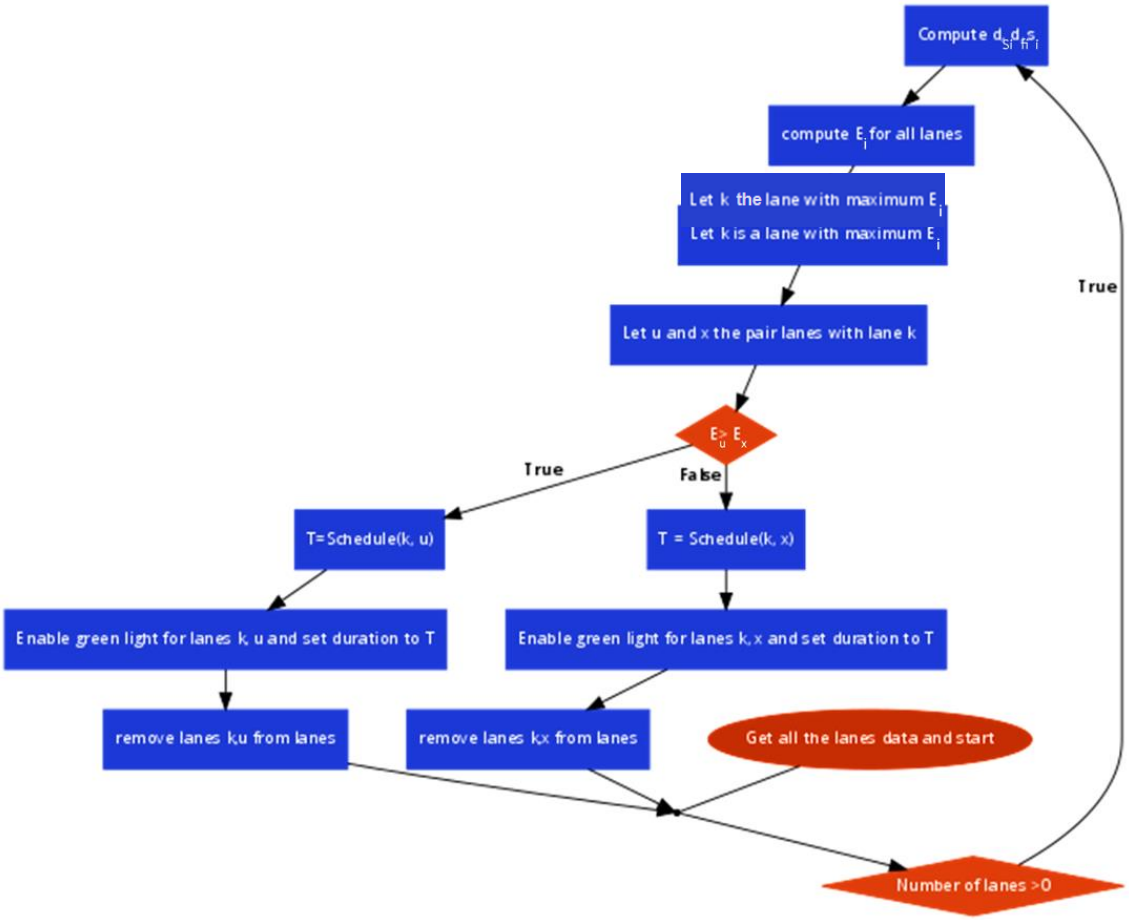

Figure 3.5: Flowchart of ATLC Algorithm

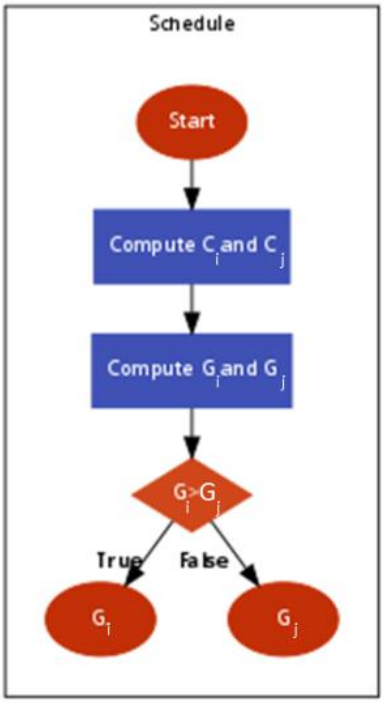

Figure 3.6: Flowchart of Schedule Function

Algorithm A illustrates the steps performed to choose the best state from Figure 3.3 that will include selecting the lane with biggest eligibility value computed in Equation (1) and its pair. In addition, a flowchart for our proposed scheme shown in Figure 3.5.

Algorithm A: Intelligent traffic light state choosing algorithm

---

Data: d_{si}: density of SA for lane i; d_{fi} : density of FA for lane i; s_i : average speed of vehicles inside of lane i; E_i : eligibility of lane i; T_{ij}: duration of green light in the selected state of traffic light with lane i and lane j as pair lanes

---

Get all the lanes and their data in the target intersection;

While number of lanes > 0 **do**

Compute $d_{si}, d_{fi} and s_i$ of each lane i;

Compute E_i for all lanes;

Let k the lane with maximumE_i;

Let u and x the pair lanes with lane k;

if $E_u > E_x$ **then**

T =Schedule (k, u);

Enable green light in (L_u, L_k) and set duration to T;

Remove lane u, k from the lanes under consideration

else

T = Schedule (k, x);

Enable green light in (L_x, L_k) and set duration to T;

Remove lane x, k from the lanes under consideration

end if

end while

We compute E_i for all lanes in the target intersection and choose the lane with maximum E-value (L_{max}). In next step, we find lanes u and v which can be paired with L_{max}, and then compute E_u and E_{vi} for all of pair lanes with L_{max}. Between lanes u and v, we choose the one with higher E value and allow it to cross the intersection, together with L_{max}.

Schedule(i, j): Schedule function for calculating duration of green light

---

Input: lane i, lane j

Output: duration of the green light

Variables: C_k: speed of traffic flow in lane k; G_k: the green light duration in lane k

---

Compute C_i and C_j according to Equation 2

Compute G_i and G_j according to Equation 3, using values computed for C_i and C_j

if $G_j > 0$

    return G_j;

else

    return G_i;

end if

In the schedule function, we can compute the time to allow all the vehicles in the SA of L_{max} to pass the intersection. But unfortunately, it is not what we exactly want. In order to take care of the vehicles in the FA that will arrive in the SA in the normal duration of the green light, we added the term E_i in the computation of G in Equation 3 permitting them to pass as well.

Chapter 4

SIMULATION ENVIRONMENT

In order to analyze the performance of the proposed scheme, a simulation web application has been developed. In this chapter, the goals of developing this simulation web application will be discussed. The simulation program has been developed by Java Script and JSON language with Canvas, HTML5 and CSS3 interface, and runs on the Linux server, with processor Intel Xeon E5-2600 v4 Processor and RAM size 16GB.

4.1 Simulation Application Scheme

In order to test the performance of our proposed algorithm, we needed a simulator to analyze it in an environment similar to a real one. We developed a simulation application in Javascript that runs on web browsers that support Javascript. Our simulation application gets the selected map file in JSON format, translates the map data to a normal Javascript array and automatically designs it in the canvas. Then it generates vehicles and places them on the map at random. A "number of vehicles to generate" option is available in this scheme and user can choose a number between 1 and 100 vehicles. Weather condition is selected randomly for the chosen date, and it affects vehicles' movement speed.

4.2 Simulation Application Interface

In our simulation application, the canvas is used to display the map on it with zoom and moving ability. A menu for options and a panel for showing results are on the left side of web page. Results panel shows when the simulation started. Figure 4.1 shows the interface of the application after the start. Options menu includes map selector,

algorithm selector, the number of vehicles, time scale and date chooser. Timescale option is the speed of running simulation.

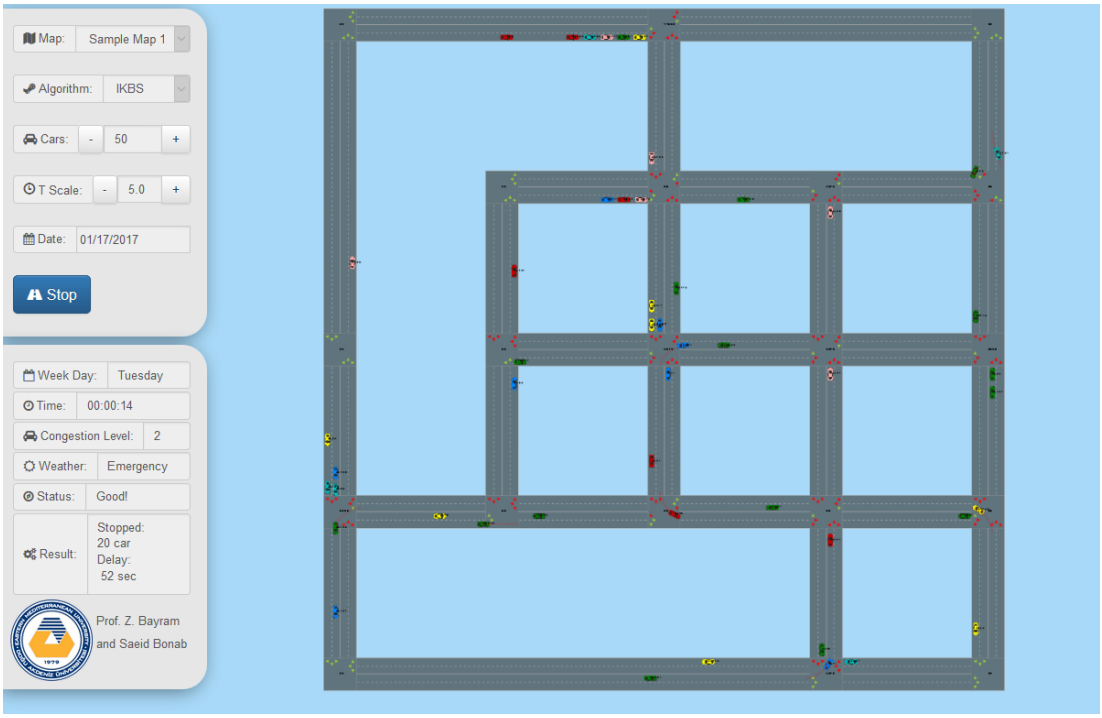

Figure 4.1: Interface of Developed Simulation Application

4.3 Simulation Application Core

As shown previously, our simulation application needs an algorithm and map to analyze. In order to have a structured map file, we put map details in a JSON file. The map file includes intersections and roads details. Our developed simulation application consists of three algorithms in the core of JavaScript code, including our proposed scheme (ATLC), intelligent traffic light controlling scheme (ITLC) [5] and the typically fixed-time scheme. For implementing all the schemes, we get all the vehicles' updated information regularly at fixed intervals and vehicles' data stored in a JSON array to be used in all the algorithms.

At the end of simulation time, one file in text format, which includes simulation results is generated automatically. Figure 4.2 shows one sample generated result file.

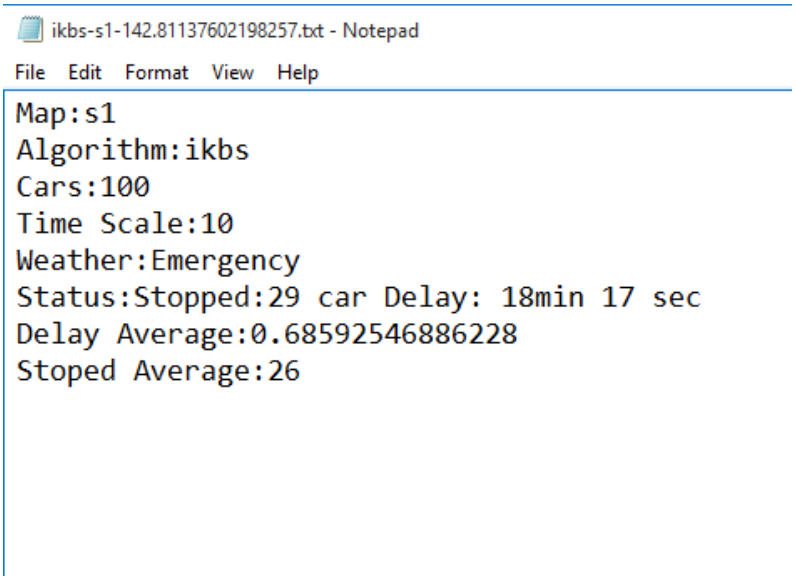

Figure 4.2: Result File Generated by Simulation Application for ATLC Scheme

In the generated file, stopped value shows a number of vehicles waiting for the green light in the last second of simulation. The car delay value is sum of delay times for all the vehicles for all the simulatin time. The delay average shows the average of the delay time of all the vehicles for the duration of the simulation and it can be calculated by using Equation 4. In addition, the stopped average shows an average number of vehicles waiting behind a traffic light in every second.

$$\text{Delay average} = \frac{\sum_{car\_id=1}^{n} \sum_{stop\_id=1}^{\#stops(car\_id)} P_{car\_id,stop\_id}}{\sum_{car\_id=1}^{n} \sum_{stop\_id=1}^{\#stops(car\_id)} 1} \quad (4)$$

where n is number of vehicles in the map, #stops is a function giving how many times a specific car has stopped and p_{ij} is the wait time for the i^{th} car at the j^{th} stop.

Chapter 5

SIMULATION RESULTS

The main goal of the proposed scheme is to decrease wait time of each vehicle behind traffic lights. The outcome of the simulations shows bright results for the proposed scheme. Three schemes have been tested by the simulation application. First one is the typically fixed time scheme where all the cycle changing duration is fixed and is generated one time (only at the beginning of the simulation) for each traffic light as a random time bigger than 3 (three) seconds and less than 30 (thirty) seconds. The second scheme is ITLC [5], and the last one is our proposed scheme (ATLC). In order to getting accurate results from each scheme, we executed the application on the same server. All the recorded results have been stored in the server as text files and for evaluating results, we calculated the average value of all experiments.

The sample map used in our simulation has been designed similar to a real city map. In addition, this map used all type of intersections such as four-way, three-way and two-way roads intersections. As shown in Figure 4.1 (page 18), in the designed map, we used 4 (four) four-way intersections, 10 (ten) three-way intersections and 5 (five) two-way intersections.

In our experiments, firstly we compared the average delay per vehicle in the total simulation time, caused by the traffic light queuing delay in intersections. We then

compared the total delay of all the vehicles on the map for the total duration of the simulation. Finally, we evaluated the results of each compared scheme.

We executed each simulation more than ten times to have more accurate results. Also, we planned to simulate in two scenarios: scenario A is in emergency weather conditions and scenario B is in normal weather conditions. The weather condition has an effect on vehicles' speed and accordingly on traffic flow speed. The following sections show the results of the two scenarios for the introduced scheme and main parameters used in our experiments is illustrated in Table 5.1.

Results for all the schemes are shown in Table 5.2 and 5.3 for scenarios A and B where simulation duration for all the tests are 30 (thirty) minutes.

Table 5.1: Simulation Main Parameters

Parameter	**Scenario A**		**Scenario B**	
	Light Traffic Flow	**Heavy Traffic Flow**	**Light Traffic Flow**	**Heavy Traffic Flow**
Number of Vehicles	50	100	50	100
Weather	Emergency	Emergency	Normal	Normal
Duration	30 minutes	30 minutes	30 minutes	30 minutes
Number of Intersections	19	19	19	19
Maximum Vehicle Speed	15	15	30	30

Table 5.2: Scenario A Results

Algorithm	Traffic Flow	Average Delay	Total Delay
Typical Fixed Time	Light	9.5388	3084
ITLS	Light	0.1574	1951
ATLC	Light	0.30288	1127
Typical Fixed Time	Heavy	8.126	3948
ITLS	Heavy	2.2502	3225
ATLC	Heavy	0.564	1585

Table 5.3: Scenario B Results

Algorithm	Traffic Flow	Average Delay	Total Delay
Typical Fixed Time	Light	5.342	1753
ITLS	Light	0.310	1651
ATLC	Light	0.131	934
Typical Fixed Time	Heavy	7.052	3416
ITLS	Heavy	2.369	2952
ATLC	Heavy	0.595	1702

5.1 Scenario A

In this scenario, we consider a situation of emergency weather condition with a maximum vehicle speed of 15. Two different cases are considered. The first one considers a map with 50 vehicles and light traffic flow in lanes, and the second case considers a map with 100 vehicles and heavy traffic flow in lanes.

Figure 5.1 shows average delay time for three mentioned schemes in light traffic flow and shows a huge difference between typically fixed time scheme and other schemes.

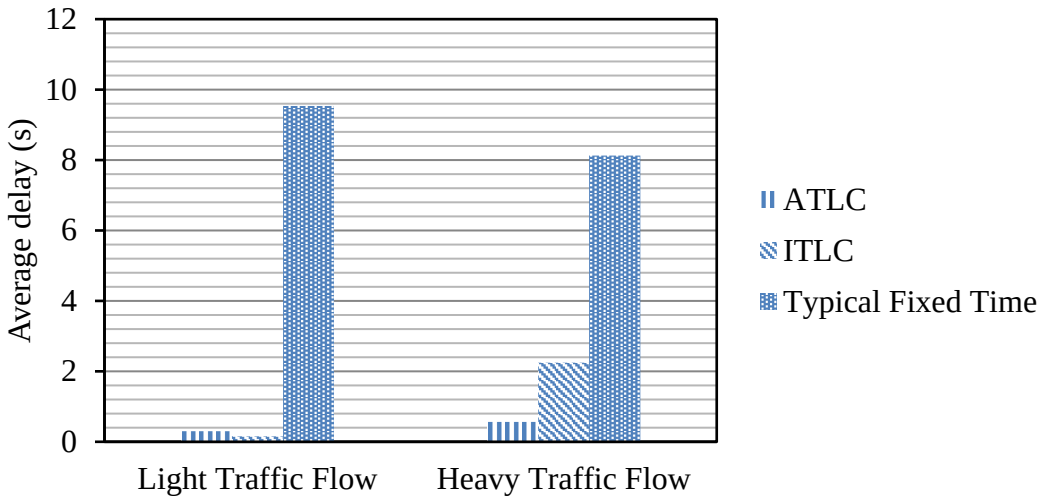

Figure 5.1: Scenario A for Average Delay Time in Seconds

Another factor evaluated in our simulation is total delay time of all the vehicles in the simulation duration. Figure 5.2 shows a big difference in total delay times of three mentioned schemes.

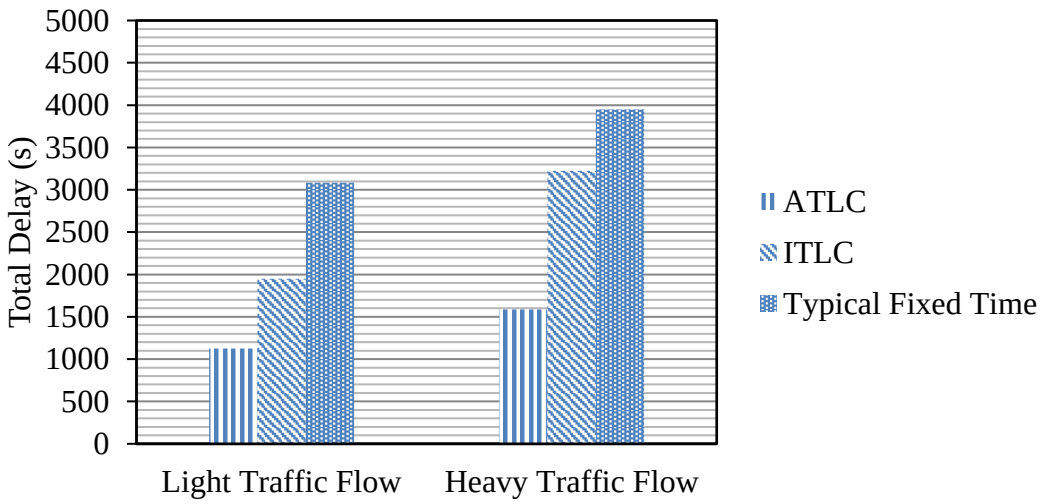

Figure 5.2: Scenario A for Total Delay Time in Seconds

From Figures 5.1 and 5.2, we can note that the difference between our proposed scheme and other mentioned schemes is significant.

Less total and average delay time show good improvement. The reduction of average delay time leads to decrease vehicles waiting time behind traffic lights and the reduction of total delay time leads to more vehicles crossing intersection which increase the throughput of vehicles flowing in the map with less time losing.

5.2 Scenario B

This scenario considers a situation of normal weather with maximum speeds of 30km/h for vehicles in the map. Outcomes of this scenario are illustrated in Figures 5.3 and 5.4.

Figure 5.3 illustrates the comparative average delay time of our proposed scheme, the ITLC and typically fixed time schemes. As shown in Figure 5.3, our introduced scheme has more favorable results, with less average delay time.

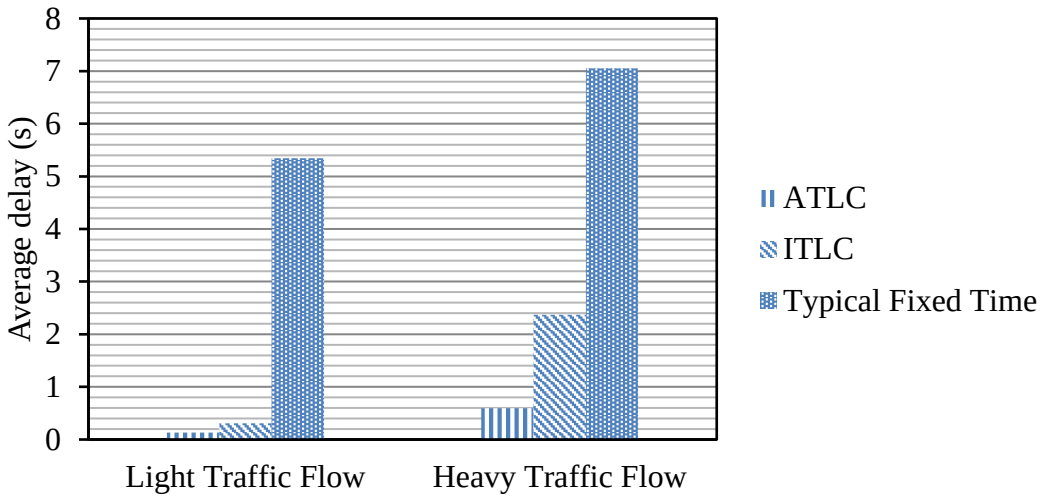

Figure 5.3: Scenario B for Average Delay Time in Seconds

We also analyzed the performance of each scheme in total delay time, as shown in Figure 5.4. We can see that ATLC has better performance in both traffic flow volumes.

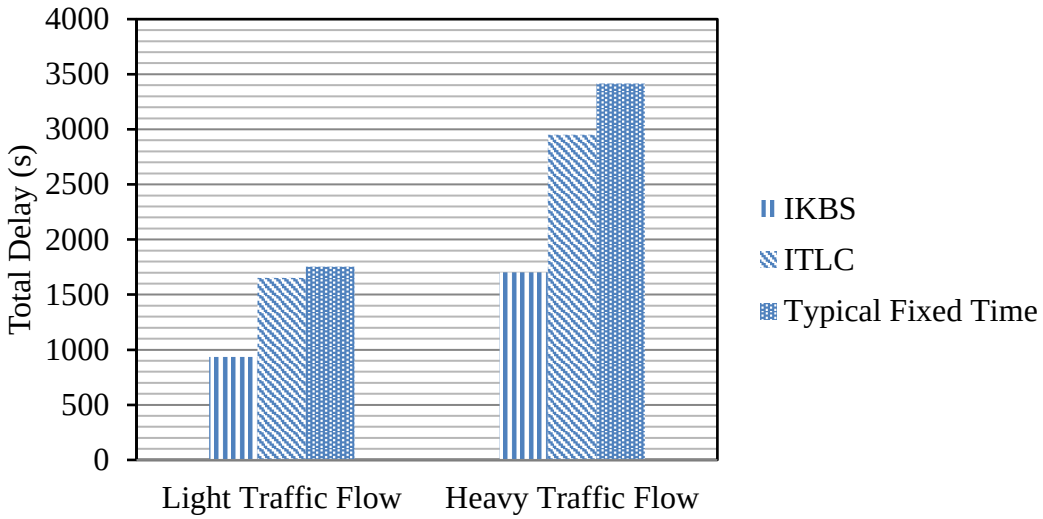

Figure 5.4: Scenario B for Total Delay Time in Seconds

5.3 Throughtput

In the throughput of the scheme, number of crossed vehicles in all the intersections are considered and calculated by dividing number of crossed vehicles from all the intersections in simulation time to simulation time in seconds. Accordingly, throughput will increase by increasing number of crossed vehicles in each intersection.

Less number of crossed vehicles will have lower throughput. All the mentioned schemes have been tested in same map and time. Results of the simultions are shown in Table 5.4 and graphical display for comparison of three schemes is displayed in Figure 5.5.

Table 5.4: Throughput of Schemes

Number of Vehicles	Vehicles Maximum Speed	Throughput (Number of crossed vehicles/second)		
		Fixed	ILTC	ATLC
50	15	11	13	15
50	30	16	18	19
100	15	29	46	51
100	30	38	42	57

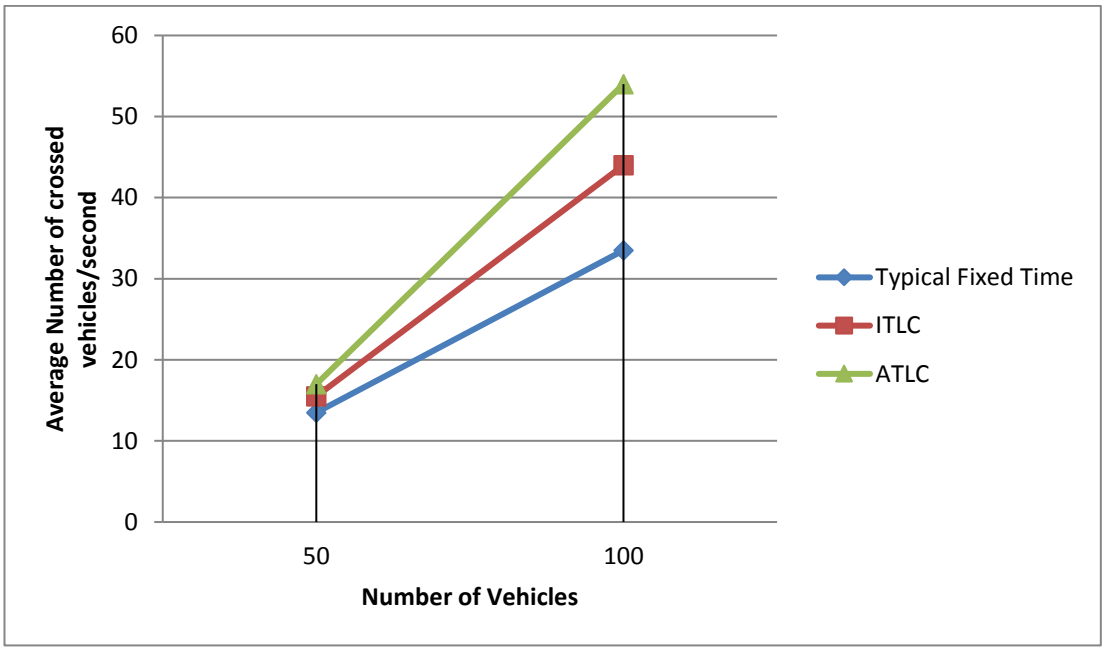

Figure 5.5: Throughput Comparison

It is clear that our proposed scheme has better throughput than the other approaches in all cases.

Chapter 6

DISCUSSION

The proposed scheme shows that by an extra survey of the traffic situation and with adequate information about transportation, we can perform a true analysis of traffic lights in intersections. In our proposed algorithm, using decision making on available data, the right lane to cross intersections is selected. The results depend on many important parameters such as congestion factor and weather condition.

In our simulations, the introduced schemes were tested for ten times in emergency weather condition and with 50 vehicles on the map. Results for average delay time for ATLC and ITLC were close to each other but there was a huge difference with typically fixed time scheme. But in total delay time, the performance of our proposed scheme was better than others. In the second experiment, we tried with 100 vehicles and emergency weather. In this condition, the performance of our scheme was the best and performance of the typical fixed time scheme was not acceptable. In the third experiment, results of the second experiment are repeated and also for the fourth experiment.

Chapter 7

CONCLUSION

We introduced an intelligent traffic controlling algorithm that considers weather condition as well as the presence of traffic affecting sites in the roads, such as schools hospitals or sport complexes. The proposed algorithm helps in adaptively assigning the green light duration to lanes without wasting time by assigning the green light to the empty lanes. Information such as roads' congestion factors and weather condition, as well as vehicles' basic travelling data assumed to be obtained from RSUs by VANET technology are used as input to our algorithm. This research highlights the importance of using VANET technology in solving traffic problems. Also, our algorithm uses near-future prediction variables for applying traffic congestion prediction in scheduling function for traffic light.

We simulated the workings of three traffic light control schemes, including our own. The simulation results show that the introduced scheme is better than the existing algorithms as it set green light duration more accurately to let more vehicles to cross the intersection in the duration of the green signal. Specifically, it resulted in lower delay times and higher throughput than the other algorithms.

In future research, the influence of pedestrians must be taken into consideration. By considering the probability of entering vehicles from one lane to another one, near-future prediction value will be more accurate, and the scheme may be extended to use

yellow light as well as green light to have more control on the intersection. Through incorporation of additional traffic information and using fuzzy theory, it will be possible to have more accurate timing of the traffic lights. Furthermore, using parameters like rush hours and roads width can enhance our results. Further research is also needed on long-term future traffic congestion prediction.

REFERENCES

[1] *M. S. Shirazi and B. T. Morris, "Looking at Intersections: A Survey of Intersection Monitoring, Behavior and Safety Analysis of Recent Studies," in* IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 1, pp. 4-24, Jan. 2017. doi: 10.1109/TITS.2016.2568920*

[2] M. Elhadef, "An Adaptable in VANETs-Based Intersection Traffic Control Algorithm," *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, Liverpool, 2015, pp. 2387-2392. doi: 10.1109/CIT/IUCC/DASC/PICOM.2015.352

[3] H. Ghaffarian, M. Fathy and M. Soryani, "Vehicular ad hoc networks enabled traffic controller for removing traffic lights in isolated intersections based on integer linear programming," in *IET Intelligent Transport Systems*, vol. 6, no. 2, pp. 115-123, June 2012. doi: 10.1049/iet-its.2010.0207

[4] Y. Xu, J. Wang, T. Liu, W. Yu and J. Xu, "Detecting Urban Road Condition and Disseminating Traffic Information by VANETs," 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and Its Associated Workshops(UIC-ATC-ScalCom), Beijing, 2015, pp. 93-98. doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.37

[5] M. Bani Younes and A. Boukerche, "An Intelligent Traffic Light scheduling algorithm through VANETs," 39th Annual IEEE Conference on Local Computer Networks Workshops, *Edmonton*, AB, 2014, pp. 637-642.doi: 10.1109/LCNW.2014.6927714

[6] E. Shaghaghi, A. Jalooli, R. Aboki, A. Marefat and R. M. Noor, "Intelligent traffic signal control for urban central using Vehicular Ad-Hoc Network," 2014 IEEE Asia Pacific Conference on Wireless and Mobile, Bali, 2014, pp. 281-286. doi: 10.1109/APWiMob.2014.6920297

[7] Zhende Xiao, Zhu Xiao, Dong Wang and Xiaohong Li, "An intelligent traffic light control approach for reducing vehicles CO2 emissions in VANET," 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery(FSKD), Zhangjiajie, 2015, pp. 2070-2075.doi: 10.1109/FSKD.2015.7382270

[8] M. B. Younes and A. Boukerche, "Efficient traffic congestion detection protocol for next generation VANETs," 2013 IEEE International Conference onCommunications *(ICC)*, Budapest, 2013, pp. 3764-3768. doi: 10.1109/ICC.2013.6655141

[9] J. Barros, M. Araujo and R. J. F. Rossetti, "Short-term real-time traffic prediction methods: A survey," 2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Budapest, 2015, pp. 132-139. doi: 10.1109/MTITS.2015.7223248

[10] Y. Regragui and N. Moussa, "Modelling and simulation of VANET in traffic city," 2014 5th Workshop on Codes, Cryptography and Communication Systems (WCCCS), El Jadida, 2014, pp. 73-76. doi: 10.1109/WCCCS.2014.7107923

[11] W. Münst *et al*., "Virtual traffic lights: Managing intersections in the cloud," *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, Munich, 2015, pp. 329-334. doi: 10.1109/RNDM.2015.7325248

[12] Dewang Chen, Xiaoming Liu, D. Lucas, Xiaoyan Gong and Suming Tan, "On criteria of setting intersection traffic light based on self-organizing theory," *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)*, Tucson, AZ, 2001, pp. 1346-1351 vol.2. doi: 10.1109/ICSMC.2001.973108

[13] V. Abrukov, V. Kochakov, A. Smirnov, S. Abrukov and D. Anufrieva, "Knowledge-based system is a goal and a tool for basic and applied research," *2015 9th International Conference on Application of Information and Communication Technologies (AICT)*, Rostov on Don, 2015, pp. 60-63. doi: 10.1109/ICAICT.2015.7338517

[14] X. Liang, Baohua Mao, Zhenqi Chen, Chaoyun Ma and Xujie Feng, "Modeling the lag of heading vehicle's startup at intersections with mixed traffic," *2010 Chinese Control and Decision Conference*, Xuzhou, 2010, pp. 4180-4185. doi: 10.1109/CCDC.2010.5498409

[15] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. 2010. Self-organized traffic control. In *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking* (VANET '10). ACM, New York, NY, USA, 85-90. DOI=http://dx.doi.org/10.1145/1860058.1860077

[16] C. Priemer and B. Friedrich, "A decentralized adaptive traffic signal control using V2I communication data," *2009 12th International IEEE Conference on Intelligent Transportation Systems*, St. Louis, MO, 2009, pp. 1-6. doi:10.1109/ITSC.2009.5309870

[17] Juan Yu, Peizhong Lu, Learning traffic signal phase and timing information from low-sampling rate taxi GPS trajectories, Knowledge-Based Systems, Volume 110, 15 October 2016, Pages 275-292, ISSN 0950-7051, http://dx.doi.org/10.1016/j.knosys.2016.07.036

[18] Shaoxin Yuan, Xiangmo Zhao, Yisheng An, Identification and optimization of traffic bottleneck with signal timing, Journal of Traffic and Transportation Engineering (English Edition), Volume 1, Issue 5, October 2014, Pages 353-361, ISSN:2095-7564,http://dx.doi.org/10.1016/S2095-7564(15)30281-6

[19] Rao Qian, Zhang Lun, Yang Wenchen, Zhang Meng, A Traffic Emission-saving Signal Timing Model for Urban Isolated Intersections, Procedia - Social and Behavioral Sciences, Volume 96, 6 November 2013, Pages 2404-2413, ISSN 1877-0428,http://dx.doi.org/10.1016/j.sbspro.2013.08.269

[20] Junchen Jin, Xiaoliang Ma, Iisakki Kosonen, An intelligent control system for traffic lights with simulation-based evaluation, Control Engineering Practice, Volume 58, January 2017, Pages 24-33, ISSN 0967-0661, http://dx.doi.org/10.1016/j.conengprac.2016.09.009

[21] M. M. M. Fahmy, "Neural Network Approach to variable vehicle speed limitation upon weather conditions," *2008 International Conference on Computer Engineering & Systems*, Cairo, 2008, pp. 179-184. doi:10.1109/ICCES.2008.4772992

APPENDICES

Appendix A: Simulation Environment Codes

Simulation Web Application Core Codes in main.js:

```
(function e(t,n,r){function s(o,u){if(!n[o]){if(!t[o]){var a=typeof require=="function"&&require;if(!u&&a)return a(o,!0);if(i)return i(o,!0);throw new Error("Cannot find module '"+o+"'")}var f=n[o]={exports:{}};t[o][0].call(f.exports,function(e){var n=t[o][1][e];return s(n?n:e)},f,f.exports,e,t,n,r)}return n[o].exports}var i=typeof require=="function"&&require;for(var o=0;o<r.length;o++)s(r[o]);return s})({1:[function(require,module,exports){

'use strict';

var $, DAT, Visualizer, World, settings, _;

require('./helpers');

$ = require('jquery');

_ = require('underscore');

Visualizer = require('./visualizer/visualizer');

DAT = require('dat-gui');

World = require('./model/world');

settings = require('./settings');

$(function() {

  var canvas, gui, guiVisualizer, guiWorld;

  canvas = $('<canvas />', {

    id: 'canvas'

  });

  $(document.body).append(canvas);

  window.world = new World();

  world.load();

  if (world.intersections.length === 0) {

    world.generateMap();

    world.carsNumber = 100;

  }

  window.visualizer = new Visualizer(world);
```

```
 visualizer.start();
 world.clear();
 //////////////Timer//////////////////////////////
  var hours, minutes, seconds, milliseconds, timer;
          var saved=0;


  var running = false;
function prependZero(time, length) {
        // Quick way to turn number to string is to prepend it with a string
        // Also, a quick way to turn floats to integers is to complement with 0
        time = '' + (time | 0);
        // And strings have length too. Prepend 0 until right.
        while (time.length< length) time = '0' + time;
        return time;
      }
      function setStopwatch(hours, minutes, seconds, milliseconds) {
        $("#c126").val(prependZero(hours, 2) + ":" + prependZero(minutes, 2) + ":" + prependZero(seconds, 2));
        if (parseInt(prependZero(minutes, 2)) === 3 && saved === 0) {
          saved = 1;
          reset();
          var content = "Map:" + $("#c29").val() + "\r\n" +
            "Algorithm:" + $("#c666").val() + "\r\n" +
            "Cars:" + $("#c25").val() + "\r\n" +
            "Time Scale:" + $("#c49").val() + "\r\n" +
            "Weather:" + $("#c128").val() + "\r\n" +
            "Status:" + $("#c1129").val() + "\r\n" +
            "Delay Average:" + parseFloat($("#c129").attr("data-dir-delay")) + "\r\n" +
            "Stoped Average:" + Math.round($("#c129").attr("data-dir-stop")) + "\r\n";
          var filename = $("#c666").val() + "-" + $("#c29").val() + "-" + (Math.random() * 1000 + 1) + ".txt";
          var blob = new Blob([content], {
```

```
            type: "text/plain;charset=utf-8"
        });
        saveAs(blob, filename);

    }
}
// Update time in stopwatch periodically - every 25ms
function runTimer() {
    // Using ES5 Date.now() to get current timestamp
    var startTime = Date.now();
    var prevHours = hours;
    var prevMinutes = minutes;
    var prevSeconds = seconds;
    var prevMilliseconds = milliseconds;
    timer = setInterval(function () {
        var timeElapsed = Date.now() - startTime;
        hours = (timeElapsed / 3600000) + prevHours;
        minutes = ((timeElapsed / 60000) + prevMinutes) % 60;
        seconds = ((timeElapsed / 1000) + prevSeconds) % 60;
        milliseconds = (timeElapsed + prevMilliseconds) % 1000;
        setStopwatch(hours, minutes, seconds, milliseconds);
    }, 25);
}
function run() {
    running = true;
    runTimer();
}
function pause() {
    running = false;
    clearTimeout(timer);
}
```

```
        function reset() {
            running = false;
            pause();
            hours = minutes = seconds = milliseconds = 0;
            setStopwatch(hours, minutes, seconds, milliseconds);
        }
        reset();
//////////////Start Button//////////////////////////////////
var weatherstatus=0;
        $('#btn').on('click', function() {
            if($("#c42").val()==="")
            {
                alert("Please Choose a date!");
            }
            else
            {
                if(started===1)
                {
                    started=0;
                    $('.dis').prop('disabled',false);
                    $('.dis2').show();
                    $('#stxt').text("Start");
                    $('#emlogo').show();
                    reset();
                    TweenMax.fromTo("#side2",1,{left:0,visibility:"visible",opacity:1},{left:-
480,opacity:0,onComplete:function(){$('#side2').css("visibility:hidden");},onStart:function()
{world.clear();}});
                }
                else
                {
                    started=1;
                    $('.dis').prop('disabled',true);
```

```
$('.dis2').hide();
$('#stxt').text("Stop");
var datesplit=$("#c42").val();
datesplit=datesplit.split('/');
if((parseInt(datesplit[0])+parseInt(datesplit[1])+parseInt(datesplit[2]))%4===0)
{
   weatherstatus=0;
   $('#c128').val("Sunny");
}
else
if((parseInt(datesplit[0])+parseInt(datesplit[1])+parseInt(datesplit[2]))%4===1)
{
   weatherstatus=1;
   $('#c128').val("Rainy");
}
else
if((parseInt(datesplit[0])+parseInt(datesplit[1])+parseInt(datesplit[2]))%4===2)
{
   weatherstatus=1;
   $('#c128').val("Snow");
}
else
if((parseInt(datesplit[0])+parseInt(datesplit[1])+parseInt(datesplit[2]))%4===3)
{
   weatherstatus=1;
   $('#c128').val("Emergency");
}
var days =
['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday'];

var myDate = new Date(parseInt(datesplit[2])+'-'+parseInt(datesplit[0])+'-
'+parseInt(datesplit[1]));
```

```
                var dayOfWeek = days[myDate.getDay()];

                $("#c125").val(dayOfWeek);

                 run();

                 $('#emlogo').hide();

                 TweenMax.fromTo("#side2",1,{left:-
480,visibility:"visible",opacity:0},{left:0,visibility:"visible",opacity:1,onComplete:function()
{world.load();}});

            }

          }


       });

// gui = new DAT.GUI();

// guiWorld = gui.addFolder('world');

// guiWorld.open();

// guiWorld.add(world, 'save');

// guiWorld.add(world, 'load');

// guiWorld.add(world, 'clear');

// guiWorld.add(world, 'generateMap');

// guiVisualizer = gui.addFolder('visualizer');

// guiVisualizer.open();

// guiVisualizer.add(visualizer, 'running').listen();

// guiVisualizer.add(visualizer, 'debug').listen();

// guiVisualizer.add(visualizer.zoomer, 'scale', 0.1, 2).listen();

// guiVisualizer.add(visualizer, 'timeFactor', 0.1, 10).listen();

// guiWorld.add(world, 'carsNumber').min(0).max(200).step(1).listen();

// guiWorld.add(world, 'instantSpeed').step(0.00001).listen();

// return gui.add(settings, 'lightsFlipInterval', 0, 400, 0.01).listen();

});

},{"./helpers":6,"./model/world":15,"./settings":16,"./visualizer/visualizer":24,"dat-
gui":27,"jquery":31,"underscore":32}],2:[function(require,module,exports){

'use strict';

var Curve, Segment;
```

```
require('../helpers');

Segment = require('./segment');

Curve = (function() {
  function Curve(A, B, O, Q) {
    this.A = A;
    this.B = B;
    this.O = O;
    this.Q = Q;
    this.AB = new Segment(this.A, this.B);
    this.AO = new Segment(this.A, this.O);
    this.OQ = new Segment(this.O, this.Q);
    this.QB = new Segment(this.Q, this.B);
    this._length = null;
  }
  Curve.property('length', {
    get: function() {
var i, point, pointsNumber, prevoiusPoint, _i;
      if (this._length == null) {
        pointsNumber = 10;
        prevoiusPoint = null;
        this._length = 0;
        for (i = _i = 0; 0 <= pointsNumber ? _i <= pointsNumber : _i >= pointsNumber; i = 0 <=
pointsNumber ? ++_i : --_i) {
          point = this.getPoint(i / pointsNumber);
          if (prevoiusPoint) {
            this._length += point.subtract(prevoiusPoint).length;
          }
          prevoiusPoint = point;
        }
      }
      return this._length;
```

```
    }
  });
  Curve.prototype.getPoint = function(a) {
    var p0, p1, p2, r0, r1;
    p0 = this.AO.getPoint(a);
    p1 = this.OQ.getPoint(a);
    p2 = this.QB.getPoint(a);
    r0 = (new Segment(p0, p1)).getPoint(a);
    r1 = (new Segment(p1, p2)).getPoint(a);
    return (new Segment(r0, r1)).getPoint(a);
  };
  Curve.prototype.getDirection = function(a) {
    var p0, p1, p2, r0, r1;
    p0 = this.AO.getPoint(a);
    p1 = this.OQ.getPoint(a);
    p2 = this.QB.getPoint(a);
    r0 = (new Segment(p0, p1)).getPoint(a);
    r1 = (new Segment(p1, p2)).getPoint(a);
    return (new Segment(r0, r1)).direction;
  };
  return Curve;
})();
module.exports = Curve;
},{"../helpers":6,"./segment":5}],3:[function(require,module,exports){
'use strict';
var Point, atan2, sqrt;
sqrt = Math.sqrt, atan2 = Math.atan2;
require('../helpers');
Point = (function() {
  function Point(x, y) {
    this.x = x != null ? x : 0;
```

```
    this.y = y != null ? y : 0;
  }
  Point.property('length', {
    get: function() {
return sqrt(this.x * this.x + this.y * this.y);
    }
  });

  Point.property('direction', {
    get: function() {
      return atan2(this.y, this.x);
    }
  });
  Point.property('normalized', {
    get: function() {
return new Point(this.x / this.length, this.y / this.length);
    }
  });
  Point.prototype.add = function(o) {
    return new Point(this.x + o.x, this.y + o.y);
  };
  Point.prototype.subtract = function(o) {
    return new Point(this.x - o.x, this.y - o.y);
  };
  Point.prototype.mult = function(k) {
return new Point(this.x * k, this.y * k);
  };
  Point.prototype.divide = function(k) {
    return new Point(this.x / k, this.y / k);
  };
  return Point;
```

```
})();
module.exports = Point;
},{"../helpers":6}],4:[function(require,module,exports){
'use strict';
var Point, Rect, Segment, abs, _;
abs = Math.abs;

require('../helpers');
_ = require('underscore');
Point = require('./point');
Segment = require('./segment');
Rect = (function() {
  function Rect(x, y, _width, _height) {
    this.x = x;
    this.y = y;
    this._width = _width != null ? _width : 0;
    this._height = _height != null ? _height : 0;
  }
  Rect.copy = function(rect) {
return new Rect(rect.x, rect.y, rect._width, rect._height);
  };
  Rect.prototype.toJSON = function() {
    return _.extend({}, this);
  };
  Rect.prototype.area = function() {
    return this.width() * this.height();
  };
  Rect.prototype.left = function(left) {
    if (left != null) {
      this.x = left;
    }
```

```
  return this.x;
};
Rect.prototype.right = function(right) {
  if (right != null) {
    this.x = right - this.width();
  }
  return this.x + this.width();
};
Rect.prototype.width = function(width) {
  if (width != null) {
    this._width = width;
  }
  return this._width;
};
Rect.prototype.top = function(top) {
  if (top != null) {
    this.y = top;
  }
  return this.y;
};
Rect.prototype.bottom = function(bottom) {
  if (bottom != null) {
    this.y = bottom - this.height();
  }
  return this.y + this.height();
};
Rect.prototype.height = function(height) {
  if (height != null) {
    this._height = height;
  }
  return this._height;
```

```
  };

  Rect.prototype.center = function(center) {

    if (center != null) {

      this.x = center.x - this.width() / 2;

      this.y = center.y - this.height() / 2;

    }

return new Point(this.x + this.width() / 2, this.y + this.height() / 2);

  };

  Rect.prototype.containsPoint = function(point) {

    var _ref, _ref1;

    return (this.left() <= (_ref = point.x) && _ref <= this.right()) && (this.top() <= (_ref1 =
point.y) && _ref1 <= this.bottom());

  };


  Rect.prototype.containsRect = function(rect) {

    return this.left() <= rect.left() && rect.right() <= this.right() && this.top() <= rect.top() &&
rect.bottom() <= this.bottom();

  };

  Rect.prototype.getVertices = function() {

return [new Point(this.left(), this.top()), new Point(this.right(), this.top()), new
Point(this.right(), this.bottom()), new Point(this.left(), this.bottom())];

  };

  Rect.prototype.getSide = function(i) {

    var vertices;

    vertices = this.getVertices();

return new Segment(vertices[i], vertices[(i + 1) % 4]);

  };

  Rect.prototype.getSectorId = function(point) {

    var offset;

    offset = point.subtract(this.center());

    if (offset.y <= 0 && abs(offset.x) <= abs(offset.y)) {

      return 0;
```

```
    }
    if (offset.x >= 0 && abs(offset.x) >= abs(offset.y)) {
      return 1;
    }
    if (offset.y >= 0 && abs(offset.x) <= abs(offset.y)) {
      return 2;
    }
    if (offset.x <= 0 && abs(offset.x) >= abs(offset.y)) {
      return 3;
    }
    throw Error('algorithm error');
  };
  Rect.prototype.getSector = function(point) {
    return this.getSide(this.getSectorId(point));
  };
  return Rect;
})();


module.exports = Rect;

},{"../helpers":6,"./point":3,"./segment":5,"underscore":32}],5:[function(require,module,exp
orts){

'use strict';

var Segment;

require('../helpers');

Segment = (function() {
  function Segment(source, target) {
    this.source = source;
    this.target = target;
  }
  Segment.property('vector', {
    get: function() {
```

```
      return this.target.subtract(this.source);
    }
  });
  Segment.property('length', {
    get: function() {
      return this.vector.length;
    }
  });

  Segment.property('direction', {
    get: function() {
      return this.vector.direction;
    }
  });
  Segment.property('center', {
    get: function() {
      return this.getPoint(0.5);
    }
  });
  Segment.prototype.split = function(n, reverse) {
var k, order, _i, _j, _k, _len, _ref, _ref1, _results, _results1, _results2;
    order = reverse ? (function() {
      _results = [];
for (var _i = _ref = n - 1; _ref<= 0 ? _i <= 0 : _i >= 0; _ref <= 0 ? _i++ : _i--){ _results.push(_i);
}
      return _results;
    }).apply(this) : (function() {
      _results1 = [];
for (var _j = 0, _ref1 = n - 1; 0<= _ref1 ? _j <= _ref1 : _j >= _ref1; 0 <= _ref1 ? _j++ : _j--){
_results1.push(_j); }
      return _results1;
    }).apply(this);
```

```
    _results2 = [];
    for (_k = 0, _len = order.length; _k < _len; _k++) {
      k = order[_k];
      _results2.push(this.subsegment(k / n, (k + 1) / n));
    }
    return _results2;
  };

  Segment.prototype.getPoint = function(a) {
return this.source.add(this.vector.mult(a));
  };
  Segment.prototype.subsegment = function(a, b) {
    var end, offset, start;
    offset = this.vector;
    start = this.source.add(offset.mult(a));
    end = this.source.add(offset.mult(b));
    return new Segment(start, end);
  };
  return Segment;
})();
module.exports = Segment;
},{"../helpers":6}],6:[function(require,module,exports){
'use strict';
module.exports = {};
Function.prototype.property = function(prop, desc) {
  return Object.defineProperty(this.prototype, prop, desc);
};
},{}],7:[function(require,module,exports){
'use strict';
var Car, Trajectory, max, min, random, sqrt, _;
var stopedAvg=0;
```

```
var delayAvg=0;
var maxAvg=0;
max = Math.max, min = Math.min, random = Math.random, sqrt = Math.sqrt;
require('../helpers');
_ = require('underscore');
Trajectory = require('./trajectory');
Car = (function() {
  function Car(lane, position) {
    this.id = _.uniqueId('car');
    this.color = (300 + 240 * random() | 0) % 360;
    this.ccar=Math.floor(Math.random()*6+1);
    this._speed = 0;
    this.width = 1.7;
//    this.length = 3 + 2 * random();
    this.length = 3 + 2;
    this.maxSpeed = 30;
    this.s0 = 2;
    this.timeHeadway = 1.5;
    this.maxAcceleration = 1;
    this.maxDeceleration = 3;
    this.trajectory = new Trajectory(this, lane, position);
    this.alive = true;
    this.preferedLane = null;
    this.stopped=0;
                        this.curlane=lane;
  }

  Car.property('coords', {
    get: function() {
      return this.trajectory.coords;
    }
```

```
  });
  Car.property('speed', {
    get: function() {
      return this._speed;
    },
    set: function(speed) {
      if (speed < 0) {
        speed = 0;
      }
      if (speed > this.maxSpeed) {
        speed = this.maxSpeed;
      }
      return this._speed = speed;
    }
  });
  Car.property('direction', {
    get: function() {
      return this.trajectory.direction;
    }
  });
  Car.prototype.release = function() {
    return this.trajectory.release();
  };
var alldelay=0;
var stopedcars=0;
  Car.prototype.getAcceleration = function() {
var a, b, breakGap, busyRoadCoeff, coeff, deltaSpeed, distanceGap, distanceToNextCar, freeRoadCoeff, intersectionCoeff, nextCarDistance, safeDistance, safeIntersectionDistance, timeGap, _ref;
    nextCarDistance = this.trajectory.nextCarDistance;
    distanceToNextCar = max(nextCarDistance.distance, 0);
    a = this.maxAcceleration;
```

```
    b = this.maxDeceleration;
deltaSpeed = (this.speed - ((_ref = nextCarDistance.car) != null ? _ref.speed : void 0)) || 0;
    freeRoadCoeff = Math.pow(this.speed / this.maxSpeed, 4);
    distanceGap = this.s0;
    timeGap = this.speed * this.timeHeadway;
    breakGap = this.speed * deltaSpeed / (2 * sqrt(a * b));
    safeDistance = distanceGap + timeGap + breakGap;
    busyRoadCoeff = Math.pow(safeDistance / distanceToNextCar, 2);
safeIntersectionDistance = 1 + timeGap + Math.pow(this.speed, 2) / (2 * b);
    intersectionCoeff = Math.pow(safeIntersectionDistance / this.trajectory.distanceToStopLine, 2);
    coeff = 1 - freeRoadCoeff - busyRoadCoeff - intersectionCoeff;
    if(this.speed<1 && this.stopped===0)
    {
       this.stptime=new Date();
       this.stopped=1;
       stopedcars=stopedcars+1;
    }
    else if(this.speed>1 && this.stopped===1)
    {
       this.stopped=0;
       stopedcars=stopedcars-1;
       var nowtime=new Date();
       alldelay=alldelay+Math.floor((nowtime- this.stptime)/1000 -0.02);
       delayAvg=(delayAvg+Math.floor((nowtime- this.stptime)/1000-0.0001))/2;

    }
                    stopedAvg=(stopedAvg+stopedcars-3)/2;

                    $("#c129").attr("data-dir-stop",stopedAvg);
                    $("#c129").attr("data-dir-delay",(delayAvg));
```

```
    if(alldelay<100)
    $("#c1129").val("Stopped:\n"+stopedcars+" car \nDelay:\n "+alldelay+" sec");
    else
    $("#c1129").val("Stopped:\n"+stopedcars+" car \nDelay:\n
"+Math.floor(alldelay/60)+"min "+alldelay%60+" sec");
    var vehnum=parseInt($("#c25").val());

    if(stopedcars>Math.floor(vehnum*2/3))
    { $("#c127").val(4);$("#c129").val("Emergency!");}
    else if(stopedcars>Math.floor(vehnum/2))
    {  $("#c127").val(3);$("#c129").val("Normal!");}
    else if(stopedcars>Math.floor(vehnum/3))
    {  $("#c127").val(2);$("#c129").val("Good!");}
    else
    { $("#c127").val(1);$("#c129").val("Perfect!");}
    return this.maxAcceleration * coeff;
  };
  Car.prototype.move = function(delta) {
    var acceleration, currentLane, preferedLane, step, turnNumber;
    acceleration = this.getAcceleration();
    this.speed += acceleration * delta;
    if (!this.trajectory.isChangingLanes && (this.nextLane)) {
      currentLane = this.trajectory.current.lane;
      turnNumber = currentLane.getTurnDirection(this.nextLane);
      preferedLane = (function() {
        switch (turnNumber) {
          case 0:
            return currentLane.leftmostAdjacent;
          case 2:
            return currentLane.rightmostAdjacent;
```

```
          default:
            return currentLane;
        }
      })();
      if (preferedLane !== currentLane) {
        this.trajectory.changeLane(preferedLane);
      }
    }
step = this.speed * delta + 0.5 * acceleration * Math.pow(delta, 2);
    if (this.trajectory.nextCarDistance.distance < step) {
      console.log('bad IDM');
    }
    if (this.trajectory.timeToMakeTurn(step)) {
      if (this.nextLane == null) {
        return this.alive = false;
      }
    }
    return this.trajectory.moveForward(step);
  };
  Car.prototype.pickNextRoad = function() {
    var currentLane, intersection, nextRoad, possibleRoads;
    intersection = this.trajectory.nextIntersection;
    currentLane = this.trajectory.current.lane;
    possibleRoads = intersection.roads.filter(function(x) {
      return x.target !== currentLane.road.source;
    });
    if (possibleRoads.length === 0) {
      return null;
    }
                        var chooseRandomRoad=[];
                        var alltheroads=possibleRoads;
```

```
for(var i in alltheroads)
{
	for(var j =0;j<alltheroads[i].roadWeight;j++)
	{
		chooseRandomRoad.push(alltheroads[i]);
	}
}
var sideValue=0;
for(var q in currentLane.road.target.inRoads)
{
sideValue=sideValue+parseInt(currentLane.road.target.inRoads[q].targetSideId);
}

var pRoads=[];
if(possibleRoads.length<2)
	nextRoad = possibleRoads[0];
else
{
	if(currentLane.leftAdjacent)
	{
		if(currentLane.road.targetSideId===0)
		{
			for(var g in currentLane.road.target.roads)
if(currentLane.road.target.roads[g].sourceSideId===3)
			  pRoads.push(currentLane.road.target.roads[g]);
			else
if(currentLane.road.target.roads[g].sourceSideId===2)
			  pRoads.push(currentLane.road.target.roads[g]);
			if(pRoads.length<1)
				for(var g in currentLane.road.target.roads)
```

```
if(currentLane.road.target.roads[g].sourceSideId===1)
                                        pRoads.push(currentLane.road.target.roads[g]);
                                    nextRoad = _.sample(pRoads);
                                    }
                                    else if(currentLane.road.targetSideId===1)
                                    {
                                            for(var g in currentLane.road.target.roads)

if(currentLane.road.target.roads[g].sourceSideId===0)
                                        pRoads.push(currentLane.road.target.roads[g]);
                                    else
if(currentLane.road.target.roads[g].sourceSideId===3)
                                        pRoads.push(currentLane.road.target.roads[g]);
                                    if(pRoads.length<1)
                                            for(var g in currentLane.road.target.roads)

if(currentLane.road.target.roads[g].sourceSideId===2)
                                        pRoads.push(currentLane.road.target.roads[g]);
                                    nextRoad = _.sample(pRoads);
                                    }
                                    else if(currentLane.road.targetSideId===2)
                                    {
                                            for(var g in currentLane.road.target.roads)

if(currentLane.road.target.roads[g].sourceSideId===1)
                                        pRoads.push(currentLane.road.target.roads[g]);
                                    else
if(currentLane.road.target.roads[g].sourceSideId===0)
                                        pRoads.push(currentLane.road.target.roads[g]);
                                    if(pRoads.length<1)
                                            for(var g in currentLane.road.target.roads)

if(currentLane.road.target.roads[g].sourceSideId===3)
```

```
                    pRoads.push(currentLane.road.target.roads[g]);
                  nextRoad = _.sample(pRoads);
                }
                else if(currentLane.road.targetSideId===3)
                {
                    for(var g in currentLane.road.target.roads)
        if(currentLane.road.target.roads[g].sourceSideId===2)
                    pRoads.push(currentLane.road.target.roads[g]);
                  else
if(currentLane.road.target.roads[g].sourceSideId===1)
                    pRoads.push(currentLane.road.target.roads[g]);
                    if(pRoads.length<1)
                        for(var g in currentLane.road.target.roads)
        if(currentLane.road.target.roads[g].sourceSideId===0)
                    pRoads.push(currentLane.road.target.roads[g]);
                  nextRoad = _.sample(pRoads);
                }
            }
            else
            {
                if(currentLane.road.targetSideId===0)
                {
                    for(var g in currentLane.road.target.roads)
        if(currentLane.road.target.roads[g].sourceSideId===1)
                    pRoads.push(currentLane.road.target.roads[g]);
                if(pRoads.length<1)
                        for(var g in currentLane.road.target.roads)
        if(currentLane.road.target.roads[g].sourceSideId===2)
                    pRoads.push(currentLane.road.target.roads[g]);
```

```
                    nextRoad = _.sample(pRoads);
                }
                else if(currentLane.road.targetSideId===1)
                {
                        for(var g in currentLane.road.target.roads)
if(currentLane.road.target.roads[g].sourceSideId===2)
                            pRoads.push(currentLane.road.target.roads[g]);
                if(pRoads.length<1)
                            for(var g in currentLane.road.target.roads)
if(currentLane.road.target.roads[g].sourceSideId===3)
                            pRoads.push(currentLane.road.target.roads[g]);
                    nextRoad = _.sample(pRoads);
                }
                else if(currentLane.road.targetSideId===2)
                {
                        for(var g in currentLane.road.target.roads)
if(currentLane.road.target.roads[g].sourceSideId===3)
                            pRoads.push(currentLane.road.target.roads[g]);
                if(pRoads.length<1)
                            for(var g in currentLane.road.target.roads)
if(currentLane.road.target.roads[g].sourceSideId===1)
                            pRoads.push(currentLane.road.target.roads[g]);
                    nextRoad = _.sample(pRoads);
                }
                else if(currentLane.road.targetSideId===3)
                {
                        for(var g in currentLane.road.target.roads)
if(currentLane.road.target.roads[g].sourceSideId===0)
```

```
                                        pRoads.push(currentLane.road.target.roads[g]);
                                    if(pRoads.length<1)
                                                for(var g in currentLane.road.target.roads)

            if(currentLane.road.target.roads[g].sourceSideId===2)
                                                pRoads.push(currentLane.road.target.roads[g]);
                                     nextRoad = _.sample(pRoads);
                                    }
                                }
                            }

                            return nextRoad;
  };
  Car.prototype.pickNextLane = function() {
    var laneNumber, nextRoad, turnNumber;
    if (this.nextLane) {
      throw Error('next lane is already chosen');
    }
    this.nextLane = null;
    nextRoad = this.pickNextRoad();
    if (!nextRoad) {
      return null;
    }
turnNumber = this.trajectory.current.lane.road.getTurnDirection(nextRoad);
    laneNumber = (function() {
      switch (turnNumber) {
        case 0:
          return nextRoad.lanesNumber - 1;
        case 1:
          return _.random(0, nextRoad.lanesNumber - 1);
        case 2:
```

```
        return 0;
      }
    })();
    this.nextLane = nextRoad.lanes[laneNumber];
    if (!this.nextLane) {
      throw Error('can not pick next lane');
    }
    return this.nextLane;
  };
  Car.prototype.popNextLane = function() {
    var nextLane;
    nextLane = this.nextLane;
    this.nextLane = null;
    this.preferedLane = null;
    return nextLane;
  };
  return Car;
})();
module.exports = Car;
},{"../helpers":6,"./trajectory":14,"underscore":32}],8:[function(require,module,exports){
'use strict';
var ControlSignals, random, settings,
  __bind = function(fn, me){ return function(){ return fn.apply(me, arguments); }; },
  __indexOf = [].indexOf || function(item) { for (var i = 0, l = this.length; i < l; i++) { if (i in this && this[i] === item) return i; } return -1; };

random = Math.random;

require('../helpers');

settings = require('../settings');
```

```
ControlSignals = (function() {
  function ControlSignals(intersection) {
    this.intersection = intersection;

    this.onTick = __bind(this.onTick, this);
    this.flipMultiplier = random();
    this.phaseOffset = 100 * random();
    this.time = this.phaseOffset;
    this.stateNum = 0;
  }
  ControlSignals.copy = function(controlSignals, intersection) {
    var result;
    if (controlSignals == null) {
      return new ControlSignals(intersection);
    }
    result = Object.create(ControlSignals.prototype);
    result.flipMultiplier = controlSignals.flipMultiplier;
    result.time = result.phaseOffset = controlSignals.phaseOffset;
    result.stateNum = 0;
    result.intersection = intersection;

    return result;
  };
  ControlSignals.prototype.toJSON = function() {
    var obj;
    return obj = {
      flipMultiplier: this.flipMultiplier,
      phaseOffset: this.phaseOffset
    };
  };
```

```
ControlSignals.prototype.states = [['L', '', 'L', ''], ['FR', '', 'FR', ''], ['', 'L', '', 'L'], ['', 'FR', '', 'FR']];
 ControlSignals.STATE = [
  {
   RED: 0,
   GREEN: 1
  }
 ];
 ControlSignals.property('flipInterval', {
    //    get: function() {
    //      return (0.1 + 0.05 * this.flipMultiplier) * settings.lightsFlipInterval;
    //    }
    get: function() {
       var _currentLines = [];
       for (var i in this.intersection.inRoads) {

          for (var j in this.intersection.inRoads[i].lanes) {
             _currentLines.push({});
             _currentLines[(_currentLines.length - 1)].road = this.intersection.inRoads[i].id;
             _currentLines[(_currentLines.length - 1)].lane = j;
             _currentLines[(_currentLines.length - 1)].len = this.intersection.inRoads[i].lanes[j].length;
             _currentLines[(_currentLines.length - 1)].stoped = 0;
             _currentLines[(_currentLines.length - 1)].moving = 0;
             _currentLines[(_currentLines.length - 1)].targetSideId = this.intersection.inRoads[i].targetSideId;
             for (var k in this.intersection.inRoads[i].lanes[j].carsPositions) {
                if (this.intersection.inRoads[i].lanes[j].carsPositions[k].car.stopped !== 0) {
                   _currentLines[(_currentLines.length - 1)].stoped = _currentLines[(_currentLines.length - 1)].stoped + 1;
                } else {
                   _currentLines[(_currentLines.length - 1)].moving = _currentLines[(_currentLines.length - 1)].moving + 1;
```

```
                }

            }

            _currentLines[(_currentLines.length - 1)].distance = _currentLines[(_currentLines.length - 1)].stoped * 5;

            _currentLines[(_currentLines.length - 1)].dens = _currentLines[(_currentLines.length - 1)].distance / _currentLines[(_currentLines.length - 1)].len;

            _currentLines[(_currentLines.length - 1)].flowTime = 3 + (_currentLines[(_currentLines.length - 1)].distance / 3);



        }

    }
    if ($("#c666").val() === "atlc") {
        var maxDens = 0,
            maxDensNum = 0;
        for (var w in _currentLines) {
            if (_currentLines[w].dens >= maxDens) {
                maxDens = _currentLines[w].dens;
                maxDensNum = w;
            }
        }
        return _currentLines[maxDensNum].flowTime;
    } else if ($("#c666").val() === "normal") {
        return (0.1 + 0.05 * this.flipMultiplier) * settings.lightsFlipInterval;
    } else if ($("#c666").val() === "itlc") {
        var maxDens = 0,
            maxDensNum = 0;
        for (var w in _currentLines) {
            if (_currentLines[w].dens >= maxDens) {
                maxDens = _currentLines[w].dens;
                maxDensNum = w;
            }
        }
```

```
            return _currentLines[maxDensNum].flowTime;
        }
    }
});
ControlSignals.prototype._decode = function(str) { //sab signal control will come here
    var state;
    state = [0, 0, 0];
    if (__indexOf.call(str, 'L') >= 0) {
        state[0] = 1;
    }
    if (__indexOf.call(str, 'F') >= 0) {
        state[1] = 1;
    }
    if (__indexOf.call(str, 'R') >= 0) {
        state[2] = 1;
    }

    return state;
};
ControlSignals.property('state', {
    get: function() {
        var stringState, x, _i, _len, _results, stateArray;
        if ($("#c666").val() === "itlc" || $("#c666").val() === "atlc") {
            stateArray = [
                ['L', '', 'L', ''],
                ['FR', '', 'FR', ''],
                ['', 'L', '', 'L'],
                ['', 'FR', '', 'FR'],
                ['LFR', '', '', ''],
                ['', 'LFR', '', ''],
                ['', '', 'LFR', ''],
```

```
            ['', '', '', 'LFR']
          ]
          stringState = stateArray[this.stateNum % stateArray.length];
        } else {
stringState = this.states[this.stateNum % this.states.length];
        }
        if (this.intersection.roads.length <= 2) {
          stringState = ['LFR', 'LFR', 'LFR', 'LFR'];
        }
        _results = [];
        for (_i = 0, _len = stringState.length; _i < _len; _i++) {
          x = stringState[_i];
          _results.push(this._decode(x));
        }
        return _results;
      }
    });
    ControlSignals.prototype.flip = function() {
      var _currentLines = [];
      for (var i in this.intersection.inRoads) {
        for (var j in this.intersection.inRoads[i].lanes) {
          _currentLines.push({});
          _currentLines[(_currentLines.length - 1)].road = this.intersection.inRoads[i].id;
          _currentLines[(_currentLines.length - 1)].lane = j;
          _currentLines[(_currentLines.length - 1)].len =
this.intersection.inRoads[i].lanes[j].length;
          _currentLines[(_currentLines.length - 1)].stoped = 0;
          _currentLines[(_currentLines.length - 1)].moving = 0;
          _currentLines[(_currentLines.length - 1)].targetSideId =
this.intersection.inRoads[i].targetSideId;
          for (var k in this.intersection.inRoads[i].lanes[j].carsPositions) {
            if (this.intersection.inRoads[i].lanes[j].carsPositions[k].car.stopped !== 0) {
```

```
                _currentLines[(_currentLines.length - 1)].stoped = _currentLines[(_currentLines.length - 1)].stoped + 1;
            } else {
                _currentLines[(_currentLines.length - 1)].moving = _currentLines[(_currentLines.length - 1)].moving + 1;
            }
        }
        _currentLines[(_currentLines.length - 1)].distance = _currentLines[(_currentLines.length - 1)].stoped * 5;
        _currentLines[(_currentLines.length - 1)].dens = _currentLines[(_currentLines.length - 1)].distance / _currentLines[(_currentLines.length - 1)].len;
        _currentLines[(_currentLines.length - 1)].flowTime = 3 + (_currentLines[(_currentLines.length - 1)].distance / 3);
    }
}
if ($("#c666").val() === "itlc") {
    var maxDens = 0,
        maxDensNum = 0;
    for (var w in _currentLines) {
        if (_currentLines[w].dens >= maxDens) {
            maxDens = _currentLines[w].dens;
            maxDensNum = w;
        }
    }
    var dFirst = -1,
        dSecond = -1;
    if (parseInt(maxDensNum) % 2 === 0)
        dFirst = (parseInt(maxDensNum) + 1);
    else
        dFirst = (parseInt(maxDensNum) - 1);
    if ((parseInt(maxDensNum) + 4) < _currentLines.length)
        dSecond = parseInt(maxDensNum) + 4;
    else if ((parseInt(maxDensNum) - 4) >= 0)
```

```
            dSecond = parseInt(maxDensNum) - 4;
        if (dFirst !== -1 && dSecond !== -1) {
            if (_currentLines[dFirst].dens > _currentLines[dSecond].dens) {
                return this.stateNum = (_currentLines[dFirst].targetSideId + 4);
            } else {
                if (dSecond % 2 === 0) {
                    if (_currentLines[dSecond].targetSideId === 3 || _currentLines[dSecond].targetSideId === 1) return this.stateNum = 1;
                    else if (_currentLines[dSecond].targetSideId === 0 || _currentLines[dSecond].targetSideId === 2) return this.stateNum = 3;
                } else if (dSecond % 2 === 1) {
                    if (_currentLines[dSecond].targetSideId === 0 || _currentLines[dSecond].targetSideId === 2) {
                        var valid = false;
                        for (var g in this.intersection.roads)
                            if (this.intersection.roads[g].sourceSideId === ((_currentLines[maxDensNum].targetSideId + 1) % 4))
                                valid = true;
                        if (!valid)
                            return this.stateNum = 3;
                        else
                            return this.stateNum = 2;
                    } else if (_currentLines[dSecond].targetSideId === 3 || _currentLines[dSecond].targetSideId === 1) {
                        var valid = false;
                        for (var g in this.intersection.roads)
                            if (this.intersection.roads[g].sourceSideId === ((_currentLines[maxDensNum].targetSideId + 1) % 4))
                                valid = true;
                        if (!valid)
                            return this.stateNum = 1;
                        else
                            return this.stateNum = 0;
                    }
```

```
            }
        }
    } else if (dFirst !== -1) {
        return this.stateNum = (_currentLines[dFirst].targetSideId + 4);
    } else {
        return this.stateNum += 1;
    }


} else if ($("#c666").val() === "atlc") {
    var maxDens = 0,
        maxDensNum = 0;
    for (var w in _currentLines) {
        if (_currentLines[w].dens >= maxDens) {
            maxDens = _currentLines[w].dens;
            maxDensNum = w;
        }
    }
    var dFirst = -1,
        dSecond = -1;
    if (parseInt(maxDensNum) % 2 === 0)
        dFirst = (parseInt(maxDensNum) + 1);
    else
        dFirst = (parseInt(maxDensNum) - 1);

    if ((parseInt(maxDensNum) + 4) < _currentLines.length)
        dSecond = parseInt(maxDensNum) + 4;
    else if ((parseInt(maxDensNum) - 4) >= 0)
        dSecond = parseInt(maxDensNum) - 4;
```

```
        if (dFirst !== -1 && dSecond !== -1) {
            if (_currentLines[dFirst].dens > _currentLines[dSecond].dens) {
                return this.stateNum = (_currentLines[dFirst].targetSideId + 4);
            } else {
                if (dSecond % 2 === 0) {
                    if (_currentLines[dSecond].targetSideId === 3 || _currentLines[dSecond].targetSideId === 1) return this.stateNum = 1;
                    else if (_currentLines[dSecond].targetSideId === 0 || _currentLines[dSecond].targetSideId === 2) return this.stateNum = 3;
                } else if (dSecond % 2 === 1) {
                    if (_currentLines[dSecond].targetSideId === 0 || _currentLines[dSecond].targetSideId === 2) {
                        var valid = false;
                        for (var g in this.intersection.roads)
                            if (this.intersection.roads[g].sourceSideId === ((_currentLines[maxDensNum].targetSideId + 1) % 4))
                                valid = true;
                        if (!valid)
                            return this.stateNum = 3;
                        else
                            return this.stateNum = 2;
                    } else if (_currentLines[dSecond].targetSideId === 3 || _currentLines[dSecond].targetSideId === 1) {
                        var valid = false;
                        for (var g in this.intersection.roads)
                            if (this.intersection.roads[g].sourceSideId === ((_currentLines[maxDensNum].targetSideId + 1) % 4))
                                valid = true;
                        if (!valid) return this.stateNum = 1;
                        else
                            return this.stateNum = 0;
                    }
                }
```

```
            }
          } else if (dFirst !== -1) {
            return this.stateNum = (_currentLines[dFirst].targetSideId + 4);
          } else {
            return this.stateNum += 1;
          }

      } else {
          return this.stateNum += 1;
      }
  };  ControlSignals.prototype.onTick = function(delta) {
                                this.time += delta;
    if (this.time > this.flipInterval) {
      this.flip();
      return this.time -= this.flipInterval;
    }
  };
  return ControlSignals;
})();
module.exports = ControlSignals;
},{"../helpers":6,"../settings":16}],9:[function(require,module,exports){
'use strict';
var ControlSignals, Intersection, Rect, _;
require('../helpers');
_ = require('underscore');
ControlSignals = require('./control-signals');
Rect = require('../geom/rect');
Intersection = (function() {
  function Intersection(rect) {
    this.rect = rect;
    this.id = _.uniqueId('intersection');
```

```
    this.roads = [];
    this.inRoads = [];
    this.controlSignals = new ControlSignals(this);
  }
  Intersection.copy = function(intersection) {
    var result;
    intersection.rect = Rect.copy(intersection.rect);
    result = Object.create(Intersection.prototype);
    _.extend(result, intersection);
    result.roads = [];
    result.inRoads = [];
    result.controlSignals = ControlSignals.copy(result.controlSignals, result);
    return result;
  };
  Intersection.prototype.toJSON = function() {
    var obj;
    return obj = {
      id: this.id,
      rect: this.rect,
      controlSignals: this.controlSignals
    };
  };
  Intersection.prototype.update = function() {
var road, _i, _j, _len, _len1, _ref, _ref1, _results;
    _ref = this.roads;
    for (_i = 0, _len = _ref.length; _i < _len; _i++) {
      road = _ref[_i];
      road.update();
    }
    _ref1 = this.inRoads;
    _results = [];
```

```
for (_j = 0, _len1 = _ref1.length; _j < _len1; _j++) {
      road = _ref1[_j];
      _results.push(road.update());
    }
    return _results;
  };
  return Intersection;
})();
module.exports = Intersection;
},{"../geom/rect":4,"../helpers":6,"./control-
signals":8,"underscore":32}],10:[function(require,module,exports){
'use strict';
var LanePosition, _;

require('../helpers');

_ = require('underscore');
LanePosition = (function() {
  function LanePosition(car, lane, position) {
    this.car = car;
    this.position = position;
    this.id = _.uniqueId('laneposition');
    this.free = true;
    this.lane = lane;
  }
  LanePosition.property('lane', {
    get: function() {
      return this._lane;
    },
    set: function(lane) {
      this.release();
```

```
      return this._lane = lane;
    }
  });
  LanePosition.property('relativePosition', {
    get: function() {
      return this.position / this.lane.length;
    }
  });
  LanePosition.prototype.acquire = function() {
    var _ref;
if (((_ref = this.lane) != null ? _ref.addCarPosition : void 0) != null) {
      this.free = false;
      return this.lane.addCarPosition(this);
    }
  };
  LanePosition.prototype.release = function() {
    var _ref;
    if (!this.free && ((_ref = this.lane) != null ? _ref.removeCar : void 0)) {
      this.free = true;
      return this.lane.removeCar(this);
    }
  };
  LanePosition.prototype.getNext = function() {
    if (this.lane && !this.free) {
      return this.lane.getNext(this);
    }
  };
  LanePosition.property('nextCarDistance', {
    get: function() {
      var frontPosition, next, rearPosition, result;
      next = this.getNext();
```

```
      if (next) {
        rearPosition = next.position - next.car.length / 2;
        frontPosition = this.position + this.car.length / 2;
        return result = {
          car: next.car,
          distance: rearPosition - frontPosition
        };
      }
      return result = {
        car: null,
        distance: Infinity
      };
    }
  });
  return LanePosition;
})();
module.exports = LanePosition;
},{"../helpers":6,"underscore":32}],11:[function(require,module,exports){
'use strict';
var Lane, Segment, _;
require('../helpers');
_ = require('underscore');
Segment = require('../geom/segment');
Lane = (function() {
  function Lane(sourceSegment, targetSegment, road) {
    this.sourceSegment = sourceSegment;
    this.targetSegment = targetSegment;
    this.road = road;
    this.leftAdjacent = null;
    this.rightAdjacent = null;
    this.leftmostAdjacent = null;
```

```
  this.rightmostAdjacent = null;
  this.carsPositions = {};
  this.update();
}
Lane.prototype.toJSON = function() {
  var obj;
  obj = _.extend({}, this);
  delete obj.carsPositions;
  return obj;
};
Lane.property('sourceSideId', {
  get: function() {
    return this.road.sourceSideId;
  }
});
Lane.property('targetSideId', {
  get: function() {
    return this.road.targetSideId;
  }
});
Lane.property('isRightmost', {
  get: function() {
    return this === this.rightmostAdjacent;
  }
});
Lane.property('isLeftmost', {
  get: function() {
    return this === this.leftmostAdjacent;
  }
});
Lane.property('leftBorder', {
```

```
    get: function() {
return new Segment(this.sourceSegment.source, this.targetSegment.target);
    }
  });
  Lane.property('rightBorder', {
    get: function() {
      return new Segment(this.sourceSegment.target, this.targetSegment.source);
    }
  });
  Lane.prototype.update = function() {
this.middleLine = new Segment(this.sourceSegment.center, this.targetSegment.center);
    this.length = this.middleLine.length;
    return this.direction = this.middleLine.direction;
  };
  Lane.prototype.getTurnDirection = function(other) {
    return this.road.getTurnDirection(other.road);
  };
  Lane.prototype.getDirection = function() {
    return this.direction;
  };
  Lane.prototype.getPoint = function(a) {
    return this.middleLine.getPoint(a);
  };
  Lane.prototype.addCarPosition = function(carPosition) {
    if (carPosition.id in this.carsPositions) {
      throw Error('car is already here');
    }

    return this.carsPositions[carPosition.id] = carPosition;
  };
  Lane.prototype.removeCar = function(carPosition) {
```

```
    if (!(carPosition.id in this.carsPositions)) {
      throw Error('removing unknown car');
    }
    return delete this.carsPositions[carPosition.id];
  };
  Lane.prototype.getNext = function(carPosition) {
    var bestDistance, distance, id, next, o, _ref;
    if (carPosition.lane !== this) {
      throw Error('car is on other lane');
    }
    next = null;
    bestDistance = Infinity;
    _ref = this.carsPositions;
    for (id in _ref) {
      o = _ref[id];
      distance = o.position - carPosition.position;
      if (!o.free && (0 < distance && distance < bestDistance)) {
        bestDistance = distance;
        next = o;
      }
    }
    return next;
  };
  return Lane;
})();
module.exports = Lane;
},{"../geom/segment":5,"../helpers":6,"underscore":32}],12:[function(require,module,exports
){
'use strict';
var Pool;
require('../helpers');
```

```
Pool = (function() {
  function Pool(factory, pool) {
    var k, v, _ref;
    this.factory = factory;
    this.objects = {};
    if ((pool != null) && (pool.objects != null)) {
      _ref = pool.objects;
      for (k in _ref) {
        v = _ref[k];
        this.objects[k] = this.factory.copy(v);
      }
    }
  }
  Pool.prototype.toJSON = function() {
    return this.objects;
  };
  Pool.prototype.get = function(id) {
    return this.objects[id];
  };

  Pool.prototype.put = function(obj) {
    return this.objects[obj.id] = obj;
  };
  Pool.prototype.pop = function(obj) {
    var id, result, _ref;
    id = (_ref = obj.id) != null ? _ref : obj;
    result = this.objects[id];
    if (typeof result.release === "function") {
      result.release();
    }
    delete this.objects[id];
```

```
    return result;
  };
  Pool.prototype.all = function() {
    return this.objects;
  };
  Pool.prototype.clear = function() {
    return this.objects = {};
  };
  Pool.property('length', {
    get: function() {
      return Object.keys(this.objects).length;
    }
  });

  return Pool;
})();
module.exports = Pool;
},{"../helpers":6}],13:[function(require,module,exports){
'use strict';
var Lane, Road, max, min, settings, _;
min = Math.min, max = Math.max;
require('../helpers');
_ = require('underscore');
Lane = require('./lane');
settings = require('../settings');
Road = (function() {
  function Road(source, target) {
    this.source = source;
    this.target = target;
    this.id = _.uniqueId('road');
    this.lanes = [];
```

```
    this.lanesNumber = null;
    this.update();
  }

  Road.copy = function(road) {
    var result;
    result = Object.create(Road.prototype);
    _.extend(result, road);
    if (result.lanes == null) {
      result.lanes = [];
    }
    return result;
  };
  Road.prototype.toJSON = function() {
    var obj;
    return obj = {
      id: this.id,
      source: this.source.id,
      target: this.target.id
    };
  };
  Road.property('length', {
    get: function() {
return this.targetSide.target.subtract(this.sourceSide.source).length;
    }
  });
  Road.property('leftmostLane', {
    get: function() {
      return this.lanes[this.lanesNumber - 1];
    }
  });
```

```
  Road.property('rightmostLane', {
    get: function() {
      return this.lanes[0];
    }
  });
  Road.prototype.getTurnDirection = function(other) {
    var side1, side2, turnNumber;
    if (this.target !== other.source) {
      throw Error('invalid roads');
    }
    side1 = this.targetSideId;
    side2 = other.sourceSideId;
return turnNumber = (side2 - side1 - 1 + 8) % 4;
  };
  Road.prototype.update = function() {
    var i, sourceSplits, targetSplits, _base, _i, _j, _ref, _ref1, _results;
    if (!(this.source && this.target)) {
      throw Error('incomplete road');
    }
this.sourceSideId = this.source.rect.getSectorId(this.target.rect.center());
    this.sourceSide = this.source.rect.getSide(this.sourceSideId).subsegment(0.5, 1.0);
this.targetSideId = this.target.rect.getSectorId(this.source.rect.center());
    this.targetSide = this.target.rect.getSide(this.targetSideId).subsegment(0, 0.5);
    this.lanesNumber = min(this.sourceSide.length, this.targetSide.length) | 0;
    this.lanesNumber = max(2, this.lanesNumber / settings.gridSize | 0);
    sourceSplits = this.sourceSide.split(this.lanesNumber, true);
    targetSplits = this.targetSide.split(this.lanesNumber);
    if ((this.lanes == null) || this.lanes.length < this.lanesNumber) {
      if (this.lanes == null) {
        this.lanes = [];
```

```
      }

      for (i = _i = 0, _ref = this.lanesNumber - 1; 0 <= _ref ? _i <= _ref : _i >= _ref; i = 0 <= _ref ? ++_i : --_i) {

        if ((_base = this.lanes)[i] == null) {

          _base[i] = new Lane(sourceSplits[i], targetSplits[i], this);

        }

      }

    }

    _results = [];

    for (i = _j = 0, _ref1 = this.lanesNumber - 1; 0 <= _ref1 ? _j <= _ref1 : _j >= _ref1; i = 0 <= _ref1 ? ++_j : --_j) {

      this.lanes[i].sourceSegment = sourceSplits[i];

      this.lanes[i].targetSegment = targetSplits[i];

      this.lanes[i].leftAdjacent = this.lanes[i + 1];

      this.lanes[i].rightAdjacent = this.lanes[i - 1];

      this.lanes[i].leftmostAdjacent = this.lanes[this.lanesNumber - 1];

      this.lanes[i].rightmostAdjacent = this.lanes[0];

      _results.push(this.lanes[i].update());

    }

    return _results;

  };

  return Road;

})();

module.exports = Road;

},{"../helpers":6,"../settings":16,"./lane":11,"underscore":32}],14:[function(require,module,exports){

'use strict';

var Curve, LanePosition, Trajectory, max, min, _;

min = Math.min, max = Math.max;

require('../helpers');

LanePosition = require('./lane-position');

Curve = require('../geom/curve');
```

```
_ = require('underscore');
Trajectory = (function() {
  function Trajectory(car, lane, position) {
    this.car = car;
    if (position == null) {
      position = 0;
    }
this.current = new LanePosition(this.car, lane, position);
    this.current.acquire();
    this.next = new LanePosition(this.car);
    this.temp = new LanePosition(this.car);
    this.isChangingLanes = false;
  }
  Trajectory.property('lane', {
    get: function() {
      return this.temp.lane || this.current.lane;
    }
  });

  Trajectory.property('absolutePosition', {
    get: function() {
      if (this.temp.lane != null) {
        return this.temp.position;
      } else {
        return this.current.position;
      }
    }
  });
  Trajectory.property('relativePosition', {
    get: function() {
      return this.absolutePosition / this.lane.length;
```

```
  }
});
Trajectory.property('direction', {
  get: function() {
    return this.lane.getDirection(this.relativePosition);
  }
});
Trajectory.property('coords', {
  get: function() {
    return this.lane.getPoint(this.relativePosition);
  }
});
Trajectory.property('nextCarDistance', {
  get: function() {
    var a, b;
    a = this.current.nextCarDistance;
    b = this.next.nextCarDistance;
    if (a.distance < b.distance) {
      return a;
    } else {
      return b;
    }
  }
});
Trajectory.property('distanceToStopLine', {
  get: function() {
    if (!this.canEnterIntersection()) {
      return this.getDistanceToIntersection();
    }
    return Infinity;
  }
```

```
  });
  Trajectory.property('nextIntersection', {
    get: function() {
      return this.current.lane.road.target;
    }
  });
  Trajectory.property('previousIntersection', {
    get: function() {
      return this.current.lane.road.source;
    }
  });

  Trajectory.prototype.isValidTurn = function() {
    var nextLane, sourceLane, turnNumber;
    nextLane = this.car.nextLane;
    sourceLane = this.current.lane;
    if (!nextLane) {
      throw Error('no road to enter');
    }
    turnNumber = sourceLane.getTurnDirection(nextLane);
    if (turnNumber === 3) {
      throw Error('no U-turns are allowed');
    }
    if (turnNumber === 0 && !sourceLane.isLeftmost) {
throw Error('no left turns from this lane');
    }
    if (turnNumber === 2 && !sourceLane.isRightmost) {
      throw Error('no right turns from this lane');
    }
    return true;
  };
```

```
Trajectory.prototype.canEnterIntersection = function() {
  var intersection, nextLane, sideId, sourceLane, turnNumber;
  nextLane = this.car.nextLane;
  sourceLane = this.current.lane;
  if (!nextLane) {
    return true;
  }
  intersection = this.nextIntersection;
  turnNumber = sourceLane.getTurnDirection(nextLane);
  sideId = sourceLane.road.targetSideId;
  return intersection.controlSignals.state[sideId][turnNumber];
};
Trajectory.prototype.getDistanceToIntersection = function() {
  var distance;
  distance = this.current.lane.length - this.car.length / 2 - this.current.position;
  if (!this.isChangingLanes) {
    return max(distance, 0);
  } else {
    return Infinity;
  }
};
Trajectory.prototype.timeToMakeTurn = function(plannedStep) {
  if (plannedStep == null) {
    plannedStep = 0;
  }
  return this.getDistanceToIntersection() <= plannedStep;
};
Trajectory.prototype.moveForward = function(distance) {
  var gap, tempRelativePosition, _ref, _ref1;
  distance = max(distance, 0);
  this.current.position += distance;
```

```
    this.next.position += distance;
    this.temp.position += distance;
    if (this.timeToMakeTurn() && this.canEnterIntersection() && this.isValidTurn()) {
      this._startChangingLanes(this.car.popNextLane(), 0);
    }
tempRelativePosition = this.temp.position / ((_ref = this.temp.lane) != null ? _ref.length : void 0);
    gap = 2 * this.car.length;
    if (this.isChangingLanes && this.temp.position > gap && !this.current.free) {
      this.current.release();
    }
    if (this.isChangingLanes && this.next.free && this.temp.position + gap > ((_ref1 = this.temp.lane) != null ? _ref1.length : void 0)) {
      this.next.acquire();
    }
    if (this.isChangingLanes && tempRelativePosition >= 1) {
      this._finishChangingLanes();
    }
    if (this.current.lane && !this.isChangingLanes && !this.car.nextLane) {
      return this.car.pickNextLane();
    }
  };
  Trajectory.prototype.changeLane = function(nextLane) {
    var nextPosition;
    if (this.isChangingLanes) {
      throw Error('already changing lane');
    }
    if (nextLane == null) {
      throw Error('no next lane');
    }
    if (nextLane === this.lane) {
      throw Error('next lane == current lane');
```

```
    }
    if (this.lane.road !== nextLane.road) {
      throw Error('not neighbouring lanes');
    }
    nextPosition = this.current.position + 3 * this.car.length;
    if (!(nextPosition < this.lane.length)) {
      throw Error('too late to change lane');
    }
    return this._startChangingLanes(nextLane, nextPosition);
  };
  Trajectory.prototype._getIntersectionLaneChangeCurve = function() {};
  Trajectory.prototype._getAdjacentLaneChangeCurve = function() {
var control1, control2, curve, direction1, direction2, distance, p1, p2;
    p1 = this.current.lane.getPoint(this.current.relativePosition);
    p2 = this.next.lane.getPoint(this.next.relativePosition);
    distance = p2.subtract(p1).length;
direction1 = this.current.lane.middleLine.vector.normalized.mult(distance * 0.3);
    control1 = p1.add(direction1);
    direction2 = this.next.lane.middleLine.vector.normalized.mult(distance * 0.3);
    control2 = p2.subtract(direction2);
    return curve = new Curve(p1, p2, control1, control2);
  };
  Trajectory.prototype._getCurve = function() {
    return this._getAdjacentLaneChangeCurve();
  };
  Trajectory.prototype._startChangingLanes = function(nextLane, nextPosition) {
    var curve;
    if (this.isChangingLanes) {
      throw Error('already changing lane');
    }
    if (nextLane == null) {
```

```
      throw Error('no next lane');
    }
    this.isChangingLanes = true;
    this.next.lane = nextLane;
    this.next.position = nextPosition;
    curve = this._getCurve();
    this.temp.lane = curve;
    this.temp.position = 0;
    return this.next.position -= this.temp.lane.length;
  };

  Trajectory.prototype._finishChangingLanes = function() {
    if (!this.isChangingLanes) {
throw Error('no lane changing is going on');
    }
    this.isChangingLanes = false;
    this.current.lane = this.next.lane;
    this.current.position = this.next.position || 0;
    this.current.acquire();
    this.next.lane = null;
    this.next.position = NaN;
    this.temp.lane = null;
    this.temp.position = NaN;
    return this.current.lane;
  };
  Trajectory.prototype.release = function() {
    var _ref, _ref1, _ref2;
    if ((_ref = this.current) != null) {
      _ref.release();
    }
    if ((_ref1 = this.next) != null) {
```

```
      _ref1.release();
    }
    return (_ref2 = this.temp) != null ? _ref2.release() : void 0;
  };
  return Trajectory;
})();
module.exports = Trajectory;
},{"../geom/curve":2,"../helpers":6,"./lane-position":10,"underscore":32}],15:[function(require,module,exports){
'use strict';
var Car, Intersection, Pool, Rect, Road, World, random, settings, _,
  __bind = function(fn, me){ return function(){ return fn.apply(me, arguments); }; };
random = Math.random;
require('../helpers');
_ = require('underscore');
Car = require('./car');
Intersection = require('./intersection');
Road = require('./road');
Pool = require('./pool');
Rect = require('../geom/rect');
settings = require('../settings');
World = (function() {
  function World() {
    this.onTick = __bind(this.onTick, this);
    this.set({});
  }
  World.property('instantSpeed', {
    get: function() {
      var speeds;
      speeds = _.map(this.cars.all(), function(car) {
        return car.speed;
```

```
      });
      if (speeds.length === 0) {
        return 0;
      }
      return (_.reduce(speeds, function(a, b) {
        return a + b;
      })) / speeds.length;
    }
  });
  World.prototype.set = function(obj) {
    if (obj == null) {
      obj = {};
    }
    this.intersections = new Pool(Intersection, obj.intersections);
    this.roads = new Pool(Road, obj.roads);
    this.cars = new Pool(Car, obj.cars);
    this.carsNumber = 0;
    return this.time = 0;
  };
  World.prototype.save = function() {
    var data;
    data = _.extend({}, this);
    delete data.cars;
    console.log(JSON.stringify(data));
    return localStorage.world = JSON.stringify(data);
  };
  World.prototype.load = function(data) {
    var id, intersection, road, _ref, _ref1, _results;
//    data = data || localStorage.world;
//      data = data && JSON.parse(data1);
```

```
if($("#c29 :selected").val()==="s1")
{
        if($("#c666").val()==="normal")
        data = JSON.parse(JSON.stringify(data1));
        else
        data = JSON.parse(JSON.stringify(data12));
}
else
{
        if($("#c666").val()==="normal")
        data = JSON.parse(JSON.stringify(data2));
        else
        data = JSON.parse(JSON.stringify(data22));
}

  if (data == null) {
    return;
  }
  this.clear();
  this.carsNumber = data.carsNumber || 0;
  _ref = data.intersections;
  for (id in _ref) {
    intersection = _ref[id];
    this.addIntersection(Intersection.copy(intersection));
  }
  _ref1 = data.roads;
  _results = [];
  for (id in _ref1) {
    road = _ref1[id];
    road = Road.copy(road);
    road.source = this.getIntersection(road.source);
```

```
    road.target = this.getIntersection(road.target);
    _results.push(this.addRoad(road));
  }
  return _results;
};
World.prototype.clear = function() {
  return this.set({});
};
World.prototype.onTick = function(delta) {
  var car, id, intersection, _ref, _ref1, _results;
  if (delta > 1) {
    throw Error('delta > 1');
  }
  this.time += delta;
  this.refreshCars();
  _ref = this.intersections.all();
  for (id in _ref) {
    intersection = _ref[id];
    intersection.controlSignals.onTick(delta);
  }
  _ref1 = this.cars.all();
  _results = [];
  for (id in _ref1) {
    car = _ref1[id];
    car.move(delta);
    if (!car.alive) {
      _results.push(this.removeCar(car));
    } else {
      _results.push(void 0);
    }
  }
```

```
    return _results;
  };
var altopts={car:50, time:5};
  World.prototype.refreshCars = function() {
    if (this.cars.length < altopts.car) {
      this.addRandomCar();
    }
    if (this.cars.length > altopts.car) {
      return this.removeRandomCar();
    }
  };
$(document).ready(function() {
        $('#c25').on('input change', function() {
            altopts.car=$('#c25').val();
          });
       });
  World.prototype.addRoad = function(road) {
    this.roads.put(road);
    road.source.roads.push(road);
    road.target.inRoads.push(road);
    return road.update();
  };
  World.prototype.getRoad = function(id) {
    return this.roads.get(id);
  };
  World.prototype.addCar = function(car) {
    return this.cars.put(car);
  };
  World.prototype.getCar = function(id) {
    return this.cars.get(id);
  };
```

```
World.prototype.removeCar = function(car) {
  return this.cars.pop(car);
};
World.prototype.addIntersection = function(intersection) {
  return this.intersections.put(intersection);
};

World.prototype.getIntersection = function(id) {
  return this.intersections.get(id);
};
World.prototype.addRandomCar = function() {
  var lane, road;
                    //sab Weight Road Random choose
                    var chooseRandRoad=[];
                    var allroads=this.roads.all();
                    for(var i in allroads)
                    {
                              for(var j =0;j<allroads[i].roadWeight;j++)
                              {
                                        chooseRandRoad.push(allroads[i]);
                              }
                    }
  road = _.sample(chooseRandRoad);
  if (road != null) {
    lane = _.sample(road.lanes);
    if (lane != null) {
      return this.addCar(new Car(lane));
    }
  }
};
World.prototype.removeRandomCar = function() {
```

```
    var car;
    car = _.sample(this.cars.all());
    if (car != null) {
      return this.removeCar(car);
    }
  };
  return World;
})();
```

Appendix B: Simulation Input Map

The structure of Map used in simulation web application:

```
var data12 = {
   "intersections": {
      "intersection2387": {
         "id": "intersection2387",
         "rect": {
            "x": 140,
            "y": -70,
            "_width": 14,
            "_height": 14
         },
         "controlSignals": {
            "flipMultiplier": 0.7895126743189256,
            "phaseOffset": 0
         }
      },
      "intersection2388": {
         "id": "intersection2388",
         "rect": {
            "x": 0,
            "y": -140,
            "_width": 14,
            "_height": 14
         },
         "controlSignals": {
            "flipMultiplier": 0.13615069476196484,
            "phaseOffset": 0
         }
      },
      "intersection2389": {
         "id": "intersection2389",
```

```
    "rect": {
      "x": 140,
      "y": 70,
      "_width": 14,
      "_height": 14
    },
    "controlSignals": {
      "flipMultiplier": 0.9474683267482911,
      "phaseOffset": 0
    }
  },
  "intersection2390": {
    "id": "intersection2390",
    "rect": {
      "x": 70,
      "y": 70,
      "_width": 14,
      "_height": 14
    },
    "controlSignals": {
      "flipMultiplier": 0.09155359984630329,
      "phaseOffset": 0
    }
  },
  "intersection2391": {
    "id": "intersection2391",
    "rect": {
      "x": -140,
      "y": 70,
      "_width": 14,
      "_height": 14
```

```
    },
    "controlSignals": {
        "flipMultiplier": 0.0691329960857554,
        "phaseOffset": 0
    }
},
"intersection2392": {
    "id": "intersection2392",
    "rect": {
        "x": -140,
        "y": -140,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.8796114170721068,
        "phaseOffset": 0
    }
},
"intersection2393": {
    "id": "intersection2393",
    "rect": {
        "x": -70,
        "y": -70,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.6160235163379648,
        "phaseOffset": 0
    }
```

```
},
"intersection2394": {
    "id": "intersection2394",
    "rect": {
        "x": 0,
        "y": 70,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.922865464843782,
        "phaseOffset": 0
    }
},
"intersection2395": {
    "id": "intersection2395",
    "rect": {
        "x": 140,
        "y": 140,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.15296342591656198,
        "phaseOffset": 0
    }
},
"intersection2396": {
    "id": "intersection2396",
    "rect": {
        "x": 0,
```

```
        "y": -70,
        "_width": 14,
        "_height": 14
      },
      "controlSignals": {
        "flipMultiplier": 0.3866481293030064,
        "phaseOffset": 0
      }
    },
    "intersection2397": {
      "id": "intersection2397",
      "rect": {
        "x": 70,
        "y": 140,
        "_width": 14,
        "_height": 14
      },
      "controlSignals": {
        "flipMultiplier": 0.1904555574453224,
        "phaseOffset": 0
      }
    },
    "intersection2398": {
      "id": "intersection2398",
      "rect": {
        "x": 140,
        "y": 0,
        "_width": 14,
        "_height": 14
      },
      "controlSignals": {
```

```
        "flipMultiplier": 0.8128562245928359,
        "phaseOffset": 0
    }
},
"intersection2399": {
    "id": "intersection2399",
    "rect": {
        "x": 140,
        "y": -140,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.13359487390989555,
        "phaseOffset": 0
    }
},
"intersection2400": {
    "id": "intersection2400",
    "rect": {
        "x": -70,
        "y": 0,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.8081184077414592,
        "phaseOffset": 0
    }
},
"intersection2401": {
```

```
    "id": "intersection2401",
    "rect": {
        "x": -70,
        "y": 70,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.9475836194425384,
        "phaseOffset": 0
    }
},
"intersection2402": {
    "id": "intersection2402",
    "rect": {
        "x": -140,
        "y": 140,
        "_width": 14,
        "_height": 14
    },
    "controlSignals": {
        "flipMultiplier": 0.9869235643484513,
        "phaseOffset": 0
    }
},
"intersection2403": {
    "id": "intersection2403",
    "rect": {
        "x": 70,
        "y": -70,
        "_width": 14,
```

```
      "_height": 14
    },
    "controlSignals": {
      "flipMultiplier": 0.41346927770639597,
      "phaseOffset": 0
    }
  },
  "intersection2404": {
    "id": "intersection2404",
    "rect": {
      "x": -140,
      "y": 0,
      "_width": 14,
      "_height": 14
    },
    "controlSignals": {
      "flipMultiplier": 0.9368524932282858,
      "phaseOffset": 0
    }
  },
  "intersection2405": {
    "id": "intersection2405",
    "rect": {
      "x": 70,
      "y": 0,
      "_width": 14,
      "_height": 14
    },
    "controlSignals": {
      "flipMultiplier": 0.3052407068468004,
      "phaseOffset": 0
```

```
        }
    },
    "intersection2406": {
        "id": "intersection2406",
        "rect": {
            "x": 0,
            "y": 0,
            "_width": 14,
            "_height": 14
        },
        "controlSignals": {
            "flipMultiplier": 0.24830751023824482,
            "phaseOffset": 0
        }
    }
},
"roads": {
    "road2407": {
        "id": "road2407",
        "source": "intersection2404",
        "target": "intersection2392",
        "roadWeight": "4"
    },
    "road2408": {
        "id": "road2408",
        "source": "intersection2392",
        "target": "intersection2404",
        "roadWeight": "5"
    },
    "road2409": {
        "id": "road2409",
```

```
    "source": "intersection2391",
    "target": "intersection2404",
    "roadWeight": "8"
},
"road2410": {
    "id": "road2410",
    "source": "intersection2404",
    "target": "intersection2391",
    "roadWeight": "10"
},
"road2411": {
    "id": "road2411",
    "source": "intersection2402",
    "target": "intersection2391",
    "roadWeight": "5"
},
"road2412": {
    "id": "road2412",
    "source": "intersection2391",
    "target": "intersection2402",
    "roadWeight": "4"
},
"road2413": {
    "id": "road2413",
    "source": "intersection2400",
    "target": "intersection2393",
    "roadWeight": "5"
},
"road2414": {
    "id": "road2414",
    "source": "intersection2393",
```

```
    "target": "intersection2400",
    "roadWeight": "5"
},
"road2415": {
    "id": "road2415",
    "source": "intersection2401",
    "target": "intersection2400",
    "roadWeight": "5"
},
"road2416": {
    "id": "road2416",
    "source": "intersection2400",
    "target": "intersection2401",
    "roadWeight": "5"
},
"road2417": {
    "id": "road2417",
    "source": "intersection2396",
    "target": "intersection2388",
    "roadWeight": "3"
},
"road2418": {
    "id": "road2418",
    "source": "intersection2388",
    "target": "intersection2396",
    "roadWeight": "5"
},
"road2419": {
    "id": "road2419",
    "source": "intersection2406",
    "target": "intersection2396",
```

```
    "roadWeight": "5"
},
"road2420": {
    "id": "road2420",
    "source": "intersection2396",
    "target": "intersection2406",
    "roadWeight": "6"
},
"road2421": {
    "id": "road2421",
    "source": "intersection2394",
    "target": "intersection2406",
    "roadWeight": "5"
},
"road2422": {
    "id": "road2422",
    "source": "intersection2406",
    "target": "intersection2394",
    "roadWeight": "5"
},
"road2423": {
    "id": "road2423",
    "source": "intersection2405",
    "target": "intersection2403",
    "roadWeight": "7"
},
"road2424": {
    "id": "road2424",
    "source": "intersection2403",
    "target": "intersection2405",
    "roadWeight": "8"
```

```
},
"road2425": {
    "id": "road2425",
    "source": "intersection2390",
    "target": "intersection2405",
    "roadWeight": "5"
},
"road2426": {
    "id": "road2426",
    "source": "intersection2405",
    "target": "intersection2390",
    "roadWeight": "5"
},
"road2427": {
    "id": "road2427",
    "source": "intersection2397",
    "target": "intersection2390",
    "roadWeight": "5"
},
"road2428": {
    "id": "road2428",
    "source": "intersection2390",
    "target": "intersection2397",
    "roadWeight": "9"
},
"road2429": {
    "id": "road2429",
    "source": "intersection2387",
    "target": "intersection2399",
    "roadWeight": "5"
},
```

```
"road2430": {
    "id": "road2430",
    "source": "intersection2399",
    "target": "intersection2387",
    "roadWeight": "6"
},
"road2431": {
    "id": "road2431",
    "source": "intersection2398",
    "target": "intersection2387",
    "roadWeight": "5"
},
"road2432": {
    "id": "road2432",
    "source": "intersection2387",
    "target": "intersection2398",
    "roadWeight": "5"
},
"road2433": {
    "id": "road2433",
    "source": "intersection2389",
    "target": "intersection2398",
    "roadWeight": "6"
},
"road2434": {
    "id": "road2434",
    "source": "intersection2398",
    "target": "intersection2389",
    "roadWeight": "5"
},
"road2435": {
```

```
    "id": "road2435",
    "source": "intersection2395",
    "target": "intersection2389",
    "roadWeight": "5"
},
"road2436": {
    "id": "road2436",
    "source": "intersection2389",
    "target": "intersection2395",
    "roadWeight": "4"
},
"road2437": {
    "id": "road2437",
    "source": "intersection2388",
    "target": "intersection2392",
    "roadWeight": "4"
},
"road2438": {
    "id": "road2438",
    "source": "intersection2392",
    "target": "intersection2388",
    "roadWeight": "5"
},
"road2439": {
    "id": "road2439",
    "source": "intersection2399",
    "target": "intersection2388",
    "roadWeight": "5"
},
"road2440": {
    "id": "road2440",
```

```
    "source": "intersection2388",
    "target": "intersection2399",
    "roadWeight": "5"
},
"road2441": {
    "id": "road2441",
    "source": "intersection2396",
    "target": "intersection2393",
    "roadWeight": "4"
},
"road2442": {
    "id": "road2442",
    "source": "intersection2393",
    "target": "intersection2396",
    "roadWeight": "5"
},
"road2443": {
    "id": "road2443",
    "source": "intersection2403",
    "target": "intersection2396",
    "roadWeight": "7"
},
"road2444": {
    "id": "road2444",
    "source": "intersection2396",
    "target": "intersection2403",
    "roadWeight": "5"
},
"road2445": {
    "id": "road2445",
    "source": "intersection2387",
```

```
    "target": "intersection2403",
    "roadWeight": "5"
},
"road2446": {
    "id": "road2446",
    "source": "intersection2403",
    "target": "intersection2387",
    "roadWeight": "9"
},
"road2447": {
    "id": "road2447",
    "source": "intersection2406",
    "target": "intersection2400",
    "roadWeight": "5"
},
"road2448": {
    "id": "road2448",
    "source": "intersection2400",
    "target": "intersection2406",
    "roadWeight": "5"
},
"road2449": {
    "id": "road2449",
    "source": "intersection2405",
    "target": "intersection2406",
    "roadWeight": "3"
},
"road2450": {
    "id": "road2450",
    "source": "intersection2406",
    "target": "intersection2405",
```

```
    "roadWeight": "5"
},
"road2451": {
    "id": "road2451",
    "source": "intersection2398",
    "target": "intersection2405",
    "roadWeight": "5"
},
"road2452": {
    "id": "road2452",
    "source": "intersection2405",
    "target": "intersection2398",
    "roadWeight": "4"
},
"road2453": {
    "id": "road2453",
    "source": "intersection2401",
    "target": "intersection2391",
    "roadWeight": "5"
},
"road2454": {
    "id": "road2454",
    "source": "intersection2391",
    "target": "intersection2401",
    "roadWeight": "5"
},
"road2455": {
    "id": "road2455",
    "source": "intersection2394",
    "target": "intersection2401",
    "roadWeight": "2"
```

```
},
"road2456": {
    "id": "road2456",
    "source": "intersection2401",
    "target": "intersection2394",
    "roadWeight": "5"
},
"road2457": {
    "id": "road2457",
    "source": "intersection2390",
    "target": "intersection2394",
    "roadWeight": "8"
},
"road2458": {
    "id": "road2458",
    "source": "intersection2394",
    "target": "intersection2390",
    "roadWeight": "5"
},
"road2459": {
    "id": "road2459",
    "source": "intersection2389",
    "target": "intersection2390",
    "roadWeight": "6"
},
"road2460": {
    "id": "road2460",
    "source": "intersection2390",
    "target": "intersection2389",
    "roadWeight": "5"
},
```

```
        "road2461": {
            "id": "road2461",
            "source": "intersection2397",
            "target": "intersection2402",
            "roadWeight": "5"
        },
        "road2462": {
            "id": "road2462",
            "source": "intersection2402",
            "target": "intersection2397",
            "roadWeight": "4"
        },
        "road2463": {
            "id": "road2463",
            "source": "intersection2395",
            "target": "intersection2397",
            "roadWeight": "4"
        },
        "road2464": {
            "id": "road2464",
            "source": "intersection2397",
            "target": "intersection2395",
            "roadWeight": "5"
        }
    },
    "carsNumber": 100,
    "time": 11.499579999996058
};
```


Appendix C: Additional Simulation Results

Algorithm	Map	Vehicles	Duration / Weather
Typical Fixed Time	S1	50	30 Min / Emergency

id	Average Delay	Average Stopped	Delay Sumup
1	8.746	24	45m 21s
2	17.564	19	45m 32s
3	8.941	25	29m 25s
4	4.92	34	44m 41s
5	7.523	40	92m 3s
Average	9.5388	28.4	3084s

Algorithm	Map	Vehicles	Duration / Weather
Typical Fixed Time	S1	100	30 Min / Emergency

id	Average Delay	Average Stopped	Delay Sumup
1	8.905	36	93m 29 s
2	7.194	41	60m 30s
3	10.877	33	84m 31s
4	3.723	31	48m 37s
5	9.931	31	41m 56s
Average	8.126	34.4	3948s

Algorithm	Map	Vehicles	Duration / Weather
ITLS	S1	50	30 Min / Emergency

id	Average Delay	Average Stopped	Delay Sumup
1	0.064	8	39m 18s
2	0.111	7	36m 42s
3	0.047	14	28m 31s
4	0.523	7	36m 36s
5	0.042	12	21m 31s
Average	0.1574	9.6	1951s

Algorithm	Map	Vehicles	Duration / Weather
ITLS	S1	100	30 Min / Emergency

id	Average Delay	Average Stopped	Delay Sumup
1	0.116	36	71m 26s
2	4.88	17	56m 12s
3	3.157	30	47m 22s
4	2.625	29	51m 5s
5	0.473	24	42m 42s
Average	2.2502	27.2	3225s

Algorithm	Map	Vehicles	Duration / Weather
ATLC	S1	50	30 Min / Emergency

id	Average Delay	Average Stopped	Delay Sumup
1	0.0014	12	17m 2s
2	0.046	6	14m 33s
3	0.5	11	26m 5s
4	0.063	10	24m 0s
5	0.904	7	12m 17s
Average	0.30288	9.2	1127s

Algorithm	Map	Vehicles	Duration / Weather
ATLC	S1	100	30 Min / Emergency

id	Average Delay	Average Stopped	Delay Sumup
1	0.685	26	18m 17s
2	0.525	30	39 m 13s
3	0.013	28	26m 48s
4	0.25	22	31m 58s
5	1.347	23	15m 52s
Average	0.564	25.8	1585s

Algorithm	Map	Vehicles	Duration / Weather
Typical Fixed Time	S1	50	30 Min / Normal

id	Average Delay	Average Stopped	Delay Sumup
1	5.232	20	32m 14s
2	8.978	22	36m 9s
3	3.2	12	22m 1s
4	5.1	15	26m 11s
5	4.2	17	29m 31s
Average	5.342	17.2	1753s

Algorithm	Map	Vehicles	Duration / Weather
Typical Fixed Time	S1	100	30 Min / Normal

id	Average Delay	Average Stopped	Delay Sumup
1	6.76	27	61m 13 s
2	8.232	21	60m 43s
3	5.86	32	68m 12s
4	6.979	19	42m 32s
5	7.43	26	52m 2s
Average	7.0522	25	3416s

Algorithm	Map	Vehicles	Duration / Weather
ITLS	S1	50	30 Min / Normal

id	Average Delay	Average Stopped	Delay Sumup
1	0.14	11	31m 8s
2	0.092	5	32m 32s
3	0.314	7	19m 51s
4	0.872	13	29m 12s
5	0.132	9	24m 53s
Average	0.31	9	1651s

Algorithm	Map	Vehicles	Duration / Weather
ITLS	S1	100	30 Min / Normal

id	Average Delay	Average Stopped	Delay Sumup
1	3.425	28	75m 12s
2	2.1	16	46m 8s
3	2.64	22	49m 3s
4	2.14	18	39m 32s
5	1.54	12	36m 8s
Average	2.369	19.2	2952s

Algorithm	Map	Vehicles	Duration / Weather
ATLC	S1	50	30 Min / Normal

id	Average Delay	Average Stopped	Delay Sumup
1	0.07	11	15m 58s
2	0.01	5	16m 5s

3	0.04	12	18m 42s
4	0.329	5	13m 12s
5	0.21	12	13m 53s
Average	0.1318	9	934s

Algorithm	Map	Vehicles	Duration / Weather
ATLC	S1	100	30 Min / Normal

id	Average Delay	Average Stopped	Delay Sumup
1	0.62	6	22m 50s
2	0.32	10	44 m 41s
3	0.14	9	29m 0s
4	0.983	11	25m 38s
5	0.914	6	19m 43s
Average	0.5954	8.4	1702s