MODELING OF MULTI LANE HIGHWAY TRAFFIC AND ANALYSIS OF JAM FORMATION

by

Ayşe Nihan İmrem

B.S., Physics, Yıldız Technical University, 2005

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Physics

Boğaziçi University

2008

ACKNOWLEDGEMENTS

I would like first to thank and express my deep gratitude to my thesis supervisor Prof. M.Levent Kurnaz, for his endless support, his guidance me to this area and his constructive criticism, patience and the motivation he has provided during this study.

Together with Prof. M.Levent Kurnaz, I also would like to thank Prof. Ali Nezihi Bilge and Assist. Prof. Tonguç Rador for their participation in my thesis committee. It is an honor for me.

I would like to thank to Batuhan Altun from Traffic Control Center for data acquisition.

I would like to thank to Onur Albayrak for his cooperation about data analysis. My special thanks are to my family, especially to my mother Bilge İmrem and my fiance Ahmet Katırcı. Without their patience and support I could not overcome the difficulties that I face during this period.

ABSTRACT

MODELING OF MULTI LANE HIGHWAY TRAFFIC AND ANALYSIS OF JAM FORMATION

Have you ever wondered how many hours you have spent sitting in a bumper-to-bumper traffic? We wrote a simulation code for defining both the Turkish driver behavior and their effect to traffic flow. We propose a Cellular automata (CA) model for traffic flow from the European side to Asian side over the Bosphorus on the Fatih Sultan Mehmet Bridge with the aim of acquiring physically significant and realistic data for a simple virtual highway consisting of four lanes, an on-ramp, an off-ramp, and an auxiliary lane. We arrange the Nagel-Schrenkenberg single lane model as multi-lane and we add different local rules to show various traffic behavior.

Traffic research still cannot fully predict under which conditions a traffic jam may suddenly occur. The aim of this thesis is to define what causes such traffic jams from a physicist's point of view. The possible reasons for the jam formation can be the frequent lane changing, less headway and high density. These parameters are changed and analyzed for their effect on the traffic flow.

We have studied space-time diagrams showing the position of all vehicles in each time step, and "fundamental diagrams" showing the relationship between flow and density. To test the psychological aspects of the driver behavior and cultural side of the traffic flow problem, a foreign graduate student and a local graduate student approach this problem using different parameters related to our traffic cultures for the same road portion.

ÖZET

ÇOK SERÍTLÍ OTOBAN MODELLEMESÎ VE TRAFÍK TIKANIKLIĞININ ANALÍZÍ

Hiç kaç saatinizi tampon tampona ilerleyen trafikte bekleyerek geçirdiğinizi düşündünüz mü? Geniş kitlelerin problemi olan trafik tıkanıklığını araştırmak ve Türk sürücü davranışını ve trafik akışına olan etkilerini gözlemlemek amacıyla bir simulasyon yazdık. Simulasyonumuz Cellular Automata modeli kullanılarak Anadolu yakası istikametinde Fatih Sultan Mehmet Köprüsü-Kavacık TEM otobanı için fizibilitesi olmayan sanal deneyler yapmak ve data elde etmek için hazırlanmıştır.

Modelleme Nagel-Schrenkenberg'in tek şeritli otoban modeli baz alınarak çok şeritli olarak hazırlanmış ve farklı trafik davranışlarını göstermek üzere yeni kurallar eklenerek dizayn edilmiştir. Trafik araştırmaları, henüz hangi unsurların tıkanıklığa neden olabileceğini öngöremiyorlar. Bizim amacımız tıkanıklığın nedenlerini bir fizikçinin bakış açısından tespit edebilmektir. Sık şerit değiştirme, takip mesafesinin yetersiz bırakılması veya yüksek yoğunluk olası nedenler arasındadır. Bu parametreler değiştirilmiş ve akışa olan etkileri analiz edilmiştir. Analizler için konum-zaman, akış-yoğunluk ve hız-zaman grafikleri kullanılmaktadır.

Çalışmamızda ayrıca trafik hususunun psikolojik ve kültürel farklılıklarını test edebilmek amacıyla, yabancı ve yerli yüksek lisans öğrencilerinin aynı yol için farklı parametreler kullanarak yaptığı modellemeler de kıyaslanmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathbf{t}$	time
ρ	total density
$\mathbf{q}$	total flow,current,throughput
$\mathbf{v^n}$	vehicle velocity at n^{th} time step
$\mathbf{x^n}$	vehicle position at n^{th} time step
$\mathbf{a^n}$	vehicle acceleration at n^{th} time step
$\mathbf{T}$	Reaction time
τ	Delay time
$\mathbf{H}$	Headway,distance between adjacent two vehicles
$\mathbf{g}$	gap
$\mathbf{P(t)}$	probability distribution of jams of lifetime
$\mathbf{Q}(\rho)$	throughput
$\mathbf{d(mainroad)}$	main road density until 300^{th} cell
$\mathbf{d(type2)}$	density of type2 that vehicles will exit from off-ramp
$\mathbf{P(r)}$	random deceleration probability
$\mathbf{P(lc)}$	lane change probability
$\mathbf{d(onramp)}$	on-ramp density

1. INTRODUCTION

We live in a city whose population is more than 10 million people and western side of Istanbul is considered a business center. People generally work on the western side and live on the eastern side of Istanbul. The traffic therefore flows from eastern side to western side in the morning, and from western side to eastern side in the evening along the two bridges. Due to the fact that when 25 lanes come from the western side after the toll booths and the number of lanes is decreased to 4, traffic jams become a huge problem. Traffic research still cannot fully predict under which conditions a traffic jam may suddenly occur. The aim of our project is to define what causes such traffic jams, from a physicist point of view. In most countries, traffic demand exceeds the capacities of the highway system. Depending on financial and environmental reasons, traffic capacities can not be expanded, so the only way is to provide solutions about finding the optimized traffic flow for minimum loss of time.

In Chapter 2, recent and current models, analysis from the viewpoint of statistical physics, and empirical observations are explained.

In Chapter 3, details of our simulation, written with Cellular Automata model, are covered. We use the Nagel-Schreckenberg (NaSch) model [2] to simulate the traffic flowing from the European side to Asian side over the Bosphorus on the Fatih Sultan Mehmet Bridge-Kavacık highway consisting of four lanes, an on-ramp, an off-ramp, and an auxiliary lane running from the beginning of the road to after the off-ramp.

In Chapter 4, the possible reasons for the jam formation are investigated. Important parameters are the frequent lane changing, less headway and density. These parameters are changed and analyzed for their effect on the traffic flow. Even when drivers obey all the rules, jams can be formed at roads with a high density of traffic.

In Istanbul, the transportation authority makes an arrangement in order to decrease the traffic jam, by taking one lane from the direction from eastern side to western

side and adds this lane to the crowded side as an auxiliary lane during the rush hours. Another important question is the effect of the auxiliary lane which is supposed to decrease the traffic congestion. As we travel daily through this traffic jam, we have the opportunity of observing the short-comings of the auxiliary lane in easing the traffic burden.

In Chapter 5, we investigate the effects of the auxiliary lane and its position on the traffic flow.

In Chapter 6, the psychological aspects of the driver behavior and cultural side of the traffic flow problem are investigated, two scientists from different cultural background approach this problem using different parameters related to their traffic cultures. Local rules are decided by the programmers according to their perception of the traffic rules. The results are given as a comparative study of these two simulations. The difference and similarities between the two approaches can be seen clearly at the graphics of the simulations.

In Conclusion, results obtained from simulation and the important realizations from the empirical and computational observations are revised. The possible applications is covered for future research.

2. TRAFFIC FLOW MODELS

In general, modeling approach to traffic can be distinguished as microscopic, macroscopic and mesoscopic.

2.1. Macroscopic Models

Macroscopic modeling maps traffic flow as a continuous unity of fluidized vehicles. No vehicle in the traffic flow is identifiable. Traffic flow is characterized by macroscopic state values like density, volume and mean velocity. Traffic flow is treated as a continuous fluid flow in fluid dynamics. The first analogy model is initiated in 1955 by the two British scientists, Lighthill and Whitham, entitled "On kinematic waves: A theory of traffic flow on long crowded roads". This theory allowed to describe the existence of shock and density waves for the first time, shown in Figure 2.1 [3]. Prigogine and Herman attempted to define traffic flow with a kind of Boltzmann like approach to the kinetic theory of gases [5], [7]. Vehicles are considered as molecules. The difference between vehicles and molecules is that the random motion of gas molecules in three dimensional space and random motion of vehicles along the road only. Analogy and stochastic models were also applied with the aim of describing the traffic flow in 1956 by Richards [32], by Prigogine in 1959 [5] and Payne in 1971 [8].

Fundamental diagram that shows the flow density relation clearly is widely used in macroscopic approach. Navier-Stokes differential equation is also used in analogy to fluid theories by Kühne [9], Kerner [10], Konhäuser and Helbing [11].

2.1.1. Continuum flow model

The analogy of traffic flow and running water has been analyzed in 1968 by May and Keller [12] in Berkeley, and Kühne [9] in Germany. The transition from the laminar flow and turbulent draining is represented by the Reynolds number in hydrodynamics.

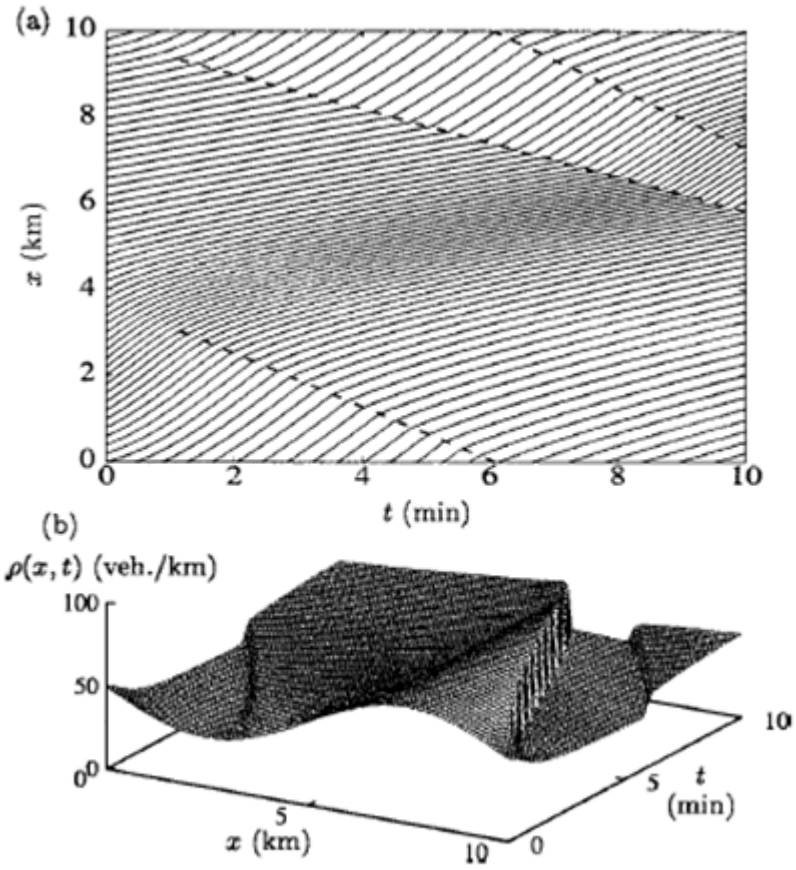


Figure 2.1. Formation of shock fronts: (a) Trajectories of each tenth vehicle and (b) corresponding spatio-temporal density plot illustrating the formation of a shock wave on a circular road according to the Lighthill-Whitham model. From Helbing [11]

.

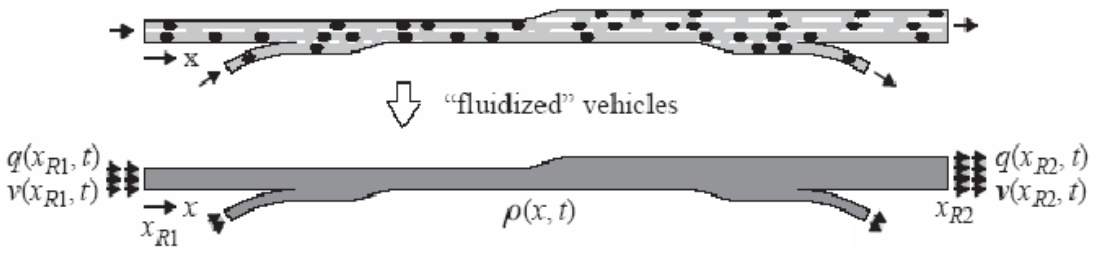


Figure 2.2. Macroscopic Model and the analogy of fluid and traffic flow [6].

- If inertia is ignored in fluid dynamical model, there would be an instant reaction, velocity adaptation. Continuity equation

$$\partial_t \rho + \partial_x q = 0 \tag{2.1}$$

where $v = \frac{q}{\rho}$ and $q = f(\rho)$, flow is a function of density.

- If inertia is considered in fluid-dynamical model, the adaptation to density needs an amount of time. Adaptation does not occur instantaneously and momentum of vehicle should be taken into account. Thus there are additional terms in acceleration:

$$a \equiv \dot{v} \equiv \partial_t.v + \partial_x.v = \frac{1}{\tau}.[V(\rho) - v] + \frac{\varsigma^2}{\rho}.\partial_x.\rho + \nu.\partial_{x^2} \tag{2.2}$$

where $V(\rho)$ is desired speed and v is actual speed. First term comes from car following model. If traffic density increase on downstream front of vehicle, it slows down, so V(gap) should be expanded as;

$$V(gap) = V(\rho) + \frac{\Delta.x}{2} = V(\rho) + \frac{1}{2}.\Delta.x.\acute{V}.\partial_x.\rho + ...$$

so gradient term of density comes into the equation. Last term represents spatial smoothing (diffusion) and space averaging effect. $\tau \to 0$ limit becomes models without inertia and the model is for instantaneous adaptation case.

2.2. Microscopic Models

In microscopic approach, each vehicle is considered as an individual particle and individual behavior of vehicle is taken into account like NaSch model. In general, microscopic models are designed with Cellular Automata. Local rules of the Cellular Automata can be variously defined according to the aim of the researchers. Car following approach model is used by Reuschel, Gazis et al. [14] [15], Montroll et al. [16], Kometani,and Sasaki [17]. In reality, vehicle motion (acceleration, deceleration) depends on the distance between the vehicle and the preceding vehicle, relative veloc-

ity of two vehicles taken with time delay. The models that the motion is independent of distance are considered as steady state solutions (hypothetical). In the "optimal velocity" model, vehicle speed is a function of the distance and speed gap between the vehicles as in Bando [18]. Wiedemann [19] added the reaction of other drivers around the vehicle and developed a psychophysical model. Kerner and Rehborn report a series of measurements they have made since 1991 on a heavily traveled stretch of highway near Frankfurt [10]. They found that traffic along this German freeway flowed in three sharply differing modes as "free flow", in which vehicles move independently of other vehicle's motion, "synchronized flow", in which speed adaptation is needed, but flow is approximately same or less than free flow and jams, in which vehicles come to at least a momentary stop. Although physicists consider second order phase transitions between the jammed and the laminar flow in response to gradual changes in average vehicle speed and traffic volume in traffic, Kerner and Rehborn claimed that the first order phase transition is the only alternative. We also investigate the validity of Boris Kerner's three phase traffic theory on Istanbul highways [10]. Two models for tracking and reconstructing spatiotemporal features of congested pattern, FOTO and ASDA models are performed without any validation of model parameters in different environmental and infrastructure situations so they are applicable for online traffic control centers for all kinds of traffic management and control.

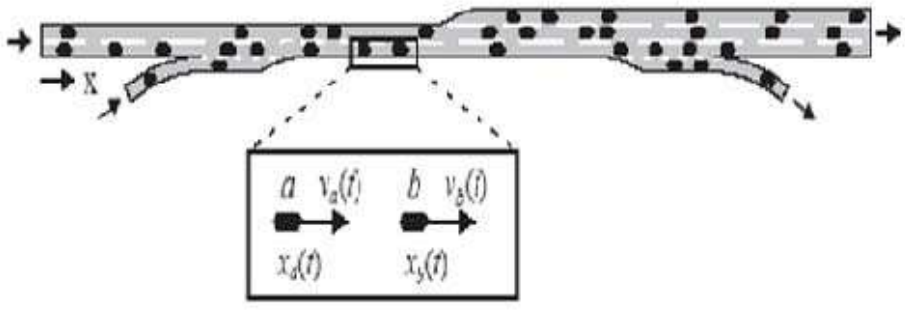


Figure 2.3. Microscopic Modeling of single highway

2.2.1. Car Following Theory

Vehicle behavior is modeled in relation to the vehicle ahead. The theory is generally used in single lane traffic model where there is no alternative from adaptation and giving reaction to the motion. $\tau = \frac{v}{g}$, where gap(g) is the front-bumper to end-bumper distance, people generally react the velocity with a delay time defined by τ,with V is gap dependent, which has been used by Newell and Whitham [20].

$$v(t+\tau) = V(gap(t), ...) \tag{2.3}$$

With Taylor expansion it becomes

$$\dot{v}(t) = a(t) = \frac{[V(gap(t) - v(t))]}{\tau} \tag{2.4}$$

$$v(t) + \tau.\dot{v}(t) + ... = V(gap(t)) \tag{2.5}$$

Stimulus-response relation where c is constant, g is gap, a is acceleration.

$$Response = sensitivity.stimulus \tag{2.6}$$

Bob Herman and coworkers experimented with the distance behavior between following vehicles, finally they describe the phantom jam which is expressed in Germany with the expression "Stau aus dem Nichts", means congestion from nowhere [7]. Actually distance oscillations between the following vehicles in high density traffic results in the phantom jam. Oscillating function of speed with respect to time is achieved at our simulation model for Turkey and it is also seen on the average velocity-time graphics obtained from traffic control center in Chapter 5. Particle hopping model is the another model, and how particle hopping models fit into the context of traffic flow theory was shown by Nagel [21] and Chowdhury et al. [22].

2.2.2. Human (Psycho-physical) Factors

One of the most interesting phenomena, found by Hoefs is that driver's sensitivity in small disturbances is larger than expanding queue of vehicles due to safety reasons. The reaction can be one of the reason of the hysteresis effect [23]. Wiedemann also apply psycho-physical factors to traffic model. If unrealistic assumptions such constant speeds or head ways turns your microscopic model into macroscopic [19].

2.2.3. Relation between Traffic and Granular flow

Friction among particles, exchange of energy among particles, interactions due to physical contact between particles in granular flow are the basic differences from traffic flow. For uncongested flow, there is no similarity between granular flow due to the lack of interaction between vehicles. Lane changing and less headway due to their desire to move, cause interaction between vehicles. Especially, bottlenecks (the decrease in the number of lanes) in the traffic flow approaches granular flow more. When vehicles are forced to merge into fewer lanes, interactions between vehicles are unavoidable. Bottlenecks are indicators of vehicular interaction. Buckley and Yagar called this event as a "capacity funnel effect" [24]. Vehicles do not collide with each other as particles in granular flow, but the expansion, compression and relaxation of headway corresponds to the deformation of particles because of collision. Catastrophe theory presents an opportunity to improve an analogy between granular and traffic flow. Better understanding of how drivers and vehicles behave at merge points is needed in order to provide improved control at the critical locations of major entrance ramps.

2.2.4. Self organized criticality and 1/f noise of traffic

Out of nowhere (phantom) jams are observed in more dense systems. Lifetime distribution P(t) always shows the power law with 3/2 exponent, in Figure 2.5.

$$P(t) \sim t^{-3/2} \tag{2.7}$$

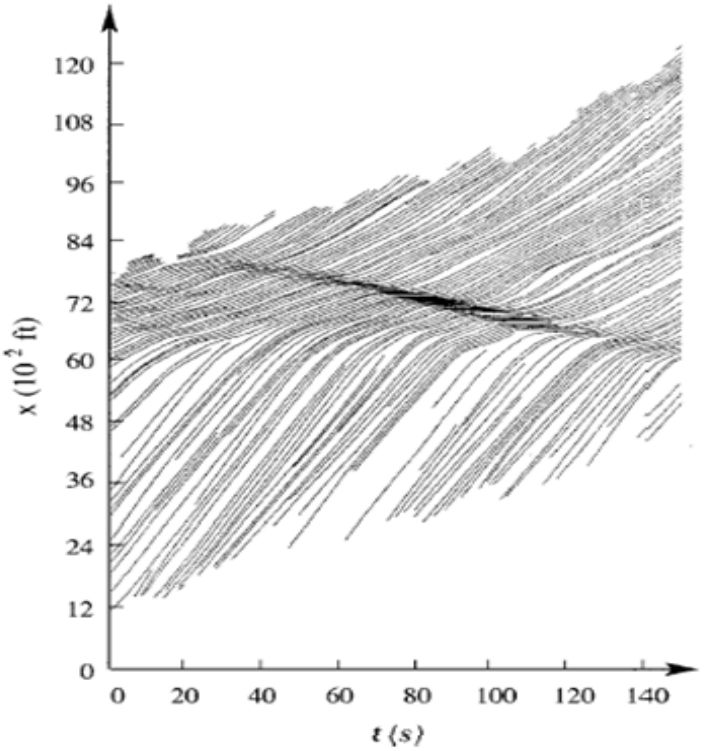


Figure 2.4. Emergence of a phantom traffic jam [25].

$$q^{\text{jam}} = \frac{\text{number of jammed vehicles}}{\text{distance between leftmost and rightmost jammed vehicles}} \tag{2.8}$$

Paczuski and Nagel analyzed validity of power spectrum theoretical assumption ($\frac{1}{f^e}$) noise (with e=1) [26]. Musha and Higuchi studied the three lanes Tomei Expressway in Japan. They recorded the transit times of passing underneath the bridge. 1/f low frequency behavior was seen in flow density graphics [27]. In Figure 2.6, the similarities between sandpile and traffic model are shown. Bak claims that complexity and self organized criticality is explained in a manner analogous to the sand pile system [28]. As you see, at low density, all cars drive with maximum speed, after passing the road capacity vehicles become dependent to the speed of preceding car and they start to adapt their speed each other. This two dimensional phase is named as synchronized flow in Kerner's three phase traffic theory [10]. At high density, long lived jams lower the throughput. Critical exponents can be obtained by analyzing the vehicles go into jam and vehicles leave the jam with random walk theory. When a vehicle reduces its speed one level with no apparent reason, even if it accelerates at next step, the following car has to decelerate, so it causes a chain reaction. Accidents cause an increase in the

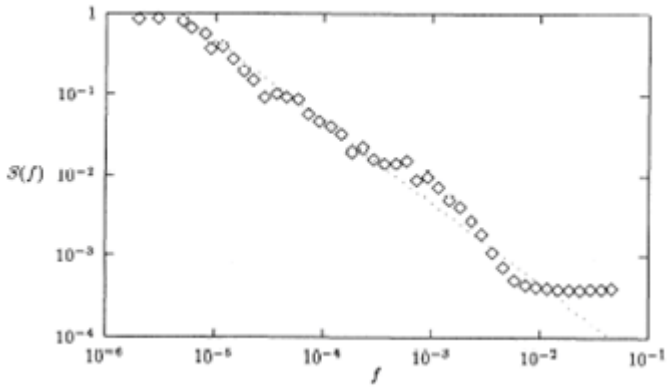
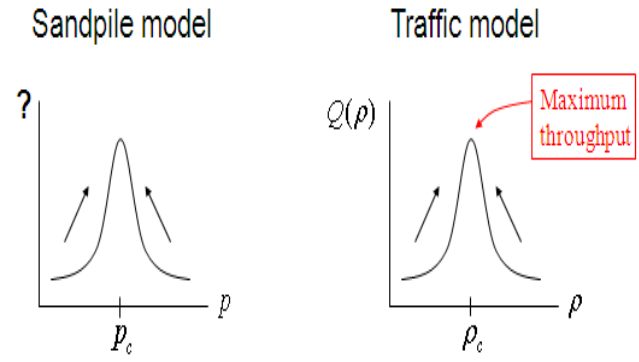


Figure 2.5. Power spectrum S(f), for a closed system of length L=$10^5, p_{free} = .00005$ [25]



	Sandpile	Traffic
Subcritical	Threshold=0	Low density
Critical	0<Threshold<∞	Critical density
Supercritical	Threshold→∞	High density

Figure 2.6. Self organized behavior similarities of sandpile and traffic model

period of deceleration process, so vehicles can not increase speed. Large disturbances create large events (jams).

2.2.5. Kerner's three phase traffic theory

There are 2 basic behavioral pattern of drivers:

- Free Flow: go ahead and try to proceed as fast as desired,
- Congested flow: try to move, adapt the speed with respect to the front and if needed change lane, even stop. In congested region, human factors play an important role, the behavior of drivers can not be predictable and models become stochastic. This freedom of drivers can be controlled by traffic authorities and rules. But actually, it is safe to say that the highest capacities are only reached when drivers are going with unsafe headway. When considered capacity, the distribution of vehicles by lanes is important.

Although two phase traffic theory, Kerner claimed that there is a phase is between congested and free flow and only first order transition is possible between phases, as shown in Figure 2.7. In Figure 2.7(a), schematic configuration of the chosen section of the highway A-5 South is shown. Characteristic dependencies of the average speed of vehicles in (b) and of the flux in (c) is shown during rush hours. In Figure 2.7(d), free flow (black points) and synchronized flow (circles) are defined on the flux density plane. Experimental points 1, 2, 3, and 4 correspond to the time 7:35, 7:36, 9:29, and 9:30, respectively. The reason behind the three phase traffic theory is the different characteristic flow named as Synchronized flow, except of free, congested flow and synchronized flow phase.

The other important empirical result is the characteristic of moving jams. According to three phase traffic theory what separates synchronized flow from jam is that jam moves backwards with constant downstream velocity, found approximately to be -16 km/hour. Kerner claimed that the downstream velocity is the same as independent of different parameters (density, flow rate) and variables (weather, road condition,

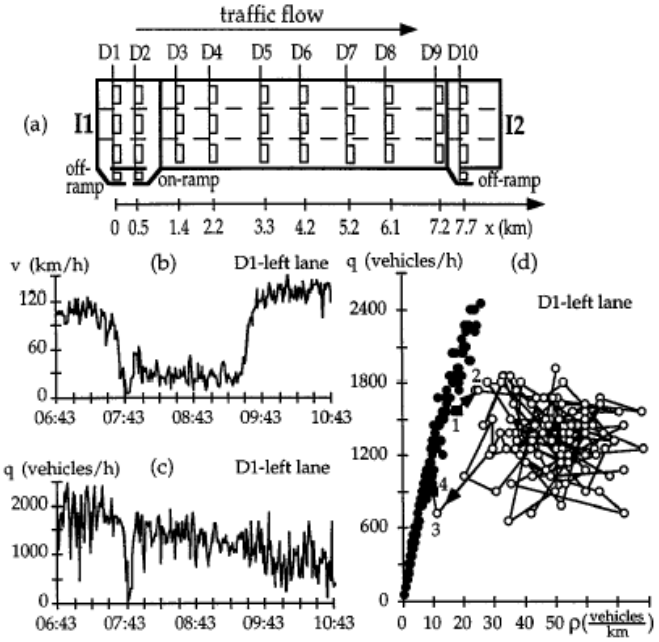


Figure 2.7. Transitions between free and synchronized traffic flow at the detectors D1 [10].

.

roadwork, personal behavior,...) and is applicable for all countries [10].

2.2.6. Nagel-Schrenkenberg Model

Definition Nagel-Schreckenberg (NaSch) model is a minimal model in that all the components of its characteristic local rules are necessary (in the order applied) to reproduce the basic features of realistic traffic [4].

In NaSch model, a one dimensional array is used and they perceive the road as one lane. A cell may be occupied or unoccupied. Occupied cells may have integer state values between (0) and maximum speed value (5). The local rule, which is applied to occupied cells, consists of four parts, which are applied in the following order:

- **Acceleration** If n^{th} vehicle (occupied cell) has a velocity less than the given

speed limit, V^n is increased by one.

$$v^n = min(v^n + 1, v_{max}) \tag{2.9}$$

- **Deceleration** v^n is the velocity of the n^{th} car and v(max) is the speed limit. If the car in front is at a distance within the range of the next movement of the n^{th} car, its speed is decreased accordingly.

$$v^n = min(v^n, d^n - 1) \tag{2.10}$$

$$d^n = x^{n+1} - x^n \tag{2.11}$$

where d^n is the gap between the n^{th} car and its predecessor

- **Randomization** If n^{th} car's speed is greater than zero, it may decrease its speed by one with some probability p.

$$v^n = max(v^n - 1, 0) \tag{2.12}$$

- **Movement** Once the new velocities are determined by the above rules, the cars are moved simultaneously.

$$x^n = x^n + v^n \tag{2.13}$$

Different traffic problems need different detailed modelings of freeway network. These patterns are analyzed with space-time diagrams. Statistical Physics is used to define some traffic flow parameters such as velocity, flow, density [1].

In analyzing results, position-time graphics showing position of each vehicle for each time step, and fundamental diagrams showing the relationship between the flow and density are used,shown in Figure 2.9.

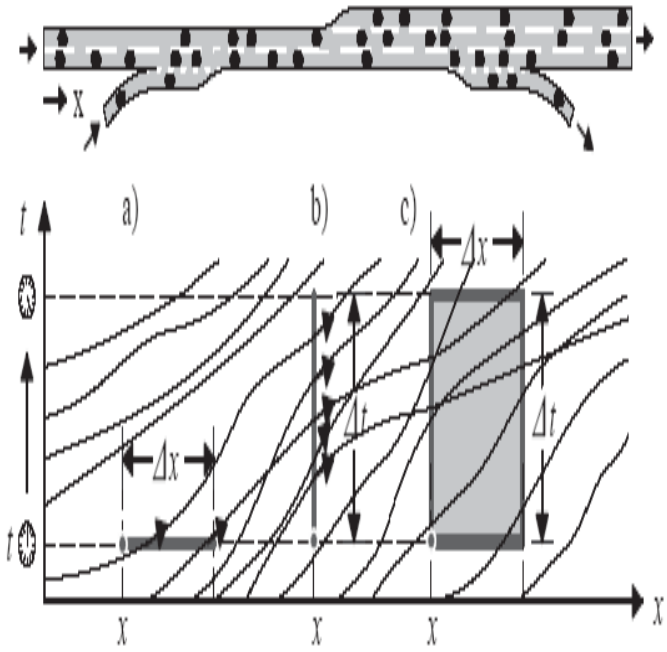


Figure 2.8. Space-Time Diagram of Freeway [31]

In Figure 2.10, for lower density, there is a linear relationship between flow and density, but for higher density, linear relationship becomes inversely proportional and the jam occurs. As you have seen above, there is a linear relationship between the flow and density before reaching the capacity of the road, after the capacity value, road overloads its capacity and vehicles have to adapt and decrease their speed, so a jam is formed. As you increase the density (number of vehicle per lane per kilometers), flow (number of vehicle per lane per hour) also increase.

2.2.7. Cellular Automata (CA) Model

The road is divided into cells and the spatial and temporal quantities are discrete. Each cell can be empty or full with a vehicle. In NaSch model the length of vehicles are considered the same and one cell's length, but vehicle length can be a parameter and a vehicle can be identified with more than one cell.

- When considered car number in 1 kilometers long road, average vehicle length is chosen as 7.5 meters.
- One timestep defines 1.2 seconds.

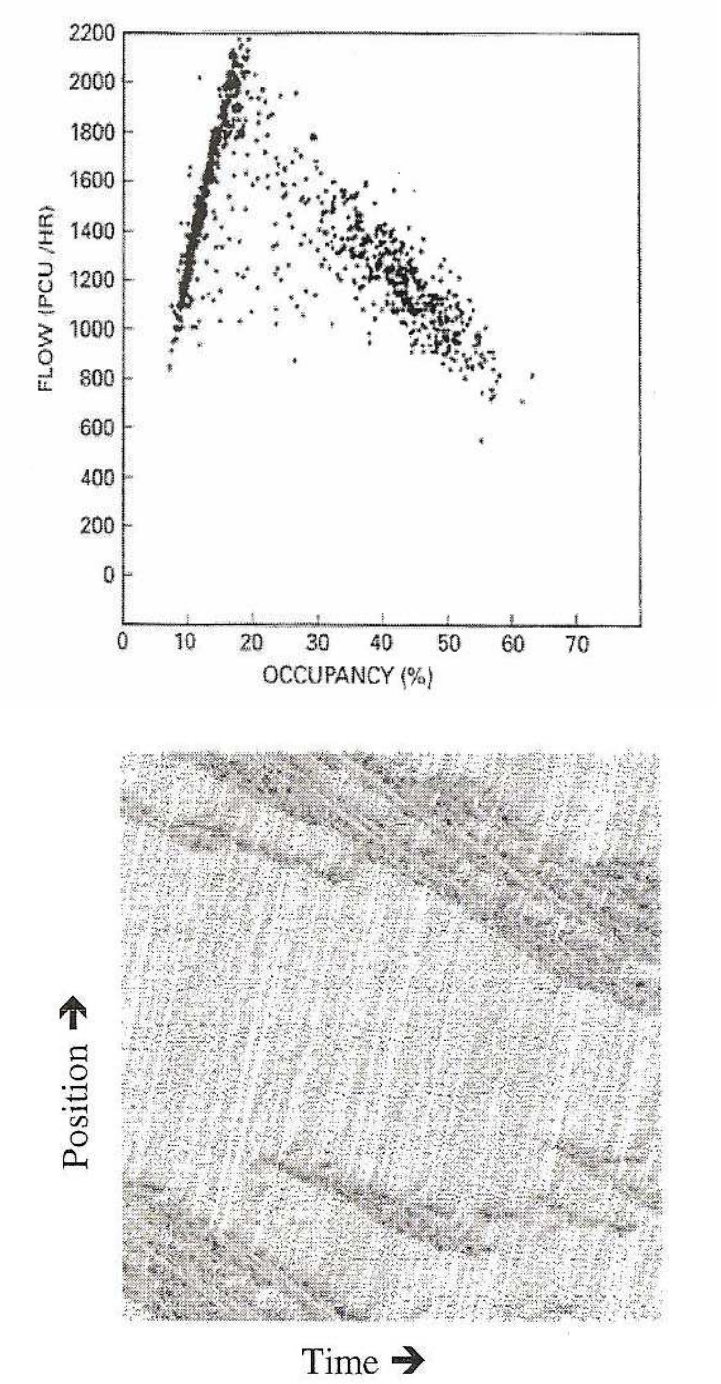


Figure 2.9. Empirical space-time and flow density Diagrams used in NaSch Model [2].

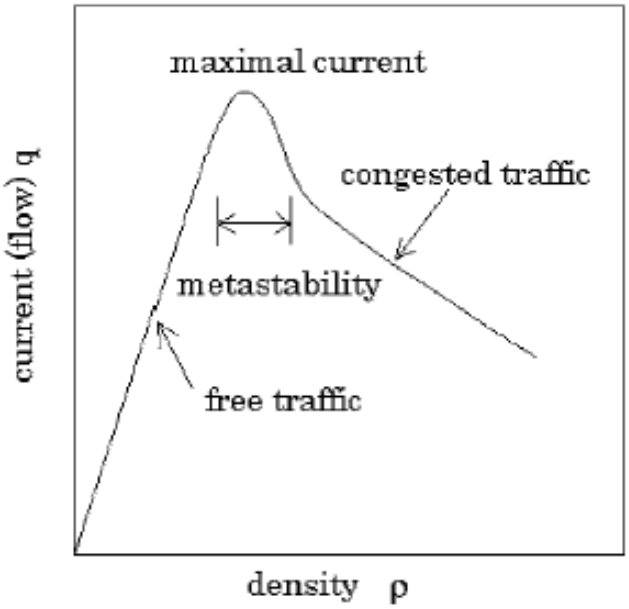


Figure 2.10. Schematic diagram of "Fundamental diagram" From [4].

- 1 cell/time step x 7.5 km/cell x (1/1.2) time step/hour = (22.5 km/h)
- Max speed 5 cells/time steps (112.5 km/h)
- The velocity defines how many "car lengths" each vehicle will move at the next time step.

Particle hopping models defines a vehicle as a particle and motion of vehicle is considered as hopping. It is similar to Cellular Automata except ASEP (Asymmetric Stochastic Exclusion Process). The difference comes from update style. While CA-184 (Wolfram) update all sites synchronously, whereas ASEP uses a random serial sequence.

Cruise Control Limit of the Stochastic Cellular Automata Model (STCA-CC) by Nagel, prevents fluctuations at free flow driving which each vehicle has maximum speed and headway is higher than maximum speed. Cruise control limit is valid when random deceleration probability, also considered in our simulation, is taken to be zero. $Gap = V^{max}$ is considered and random deceleration and braking probability is taken zero for this model, it is obviously said the model is more deterministic and laminar than stochastic CA Model. The vehicle comes to the end of the jam upstream front increases the number of vehicle in jam region by one, and vehicle which leaves the

jam downstream front, decreases number of vehicle in jam region, by one. q^{in} is the incoming flow rate to jam and q^{out} is the out coming flow rate from jam. If the motion of vehicle is thought as random walk,

- $q^{in} < q^{out}$ jams shrink and finally dissolve;
- $q^{in} > q^{out}$ jams continuously grows;
- $q^{in}=q^{out}$ jams have critical fluctuations, which can be described by critical components;

"The chess-board is the world; the pieces are the phenomena of the universe; the rules of the game are what we call the laws of Nature." Thomas Henry Huxley

2.3. Mesoscopic Modeling

The mesoscopic scale refers to the length scale at which one can reasonably discuss the properties of a material or phenomenon without having to discuss the behavior of individual atoms, and concepts of averages such as density and temperature are useful. In mesoscopic modeling of traffic flow is actually between macroscopic and microscopic modeling. K. T. Waldeer solved a spatial homogeneous vehicular traffic flow model based on a stochastic master equation of Boltzmann type in the acceleration variable is solved numerically for a special driver interaction model. The solution is done by a modified direct simulation Monte Carlo method well known in non-equilibrium gas kinetic. The velocity and acceleration distribution functions in stochastic equilibrium, mean velocity, traffic density, accelerated noise (ACN), velocity scattering and correlations between some of these variables and their car density dependence can be found by mesoscopic modeling [33].

3. OUR MODEL (MODEL II)

3.1. Parameters

A five dimensional array is used as an highway with an on-ramp, off-ramp and auxiliary lane, main road includes 500 cells and 4 lanes. The maximum speed is (5), one speed level corresponds to 22.5 km/h, one cell length is 7.5 m, which agrees with NaSch model [4]. The density of the main road is measured for four lane and auxiliary lane from the beginning of the road to the off-ramp at 300^{th} cell. The entrance probability of a new car and its type are regulated with the use of this density. The densities of the on-ramp and auxiliary lane can be changed and analyzed for different main flow densities. On-ramp begins at the 300^{th} cell and ends at the 330^{th} cell. Auxiliary lane begins at the beginning of the road and it merges into the main road at 350^{th} cell. This roughly simulates the road between FSM toll booths and Molla Gürani viaduct. The density of cars that exit from the off-ramp is identified initially and the cars are named as being of second type. Type 2 vehicles define vehicles that will exit from off-ramp and density of Type 2 vehicles is considered as 5 % of the density of the road portion from the beginning of the road, to off-ramp position (30 %) in our model. The cars of first type cars do not exit from the off-ramp. Due to the position of the auxiliary lane entrance to the main road, cars of second type are not allowed in the auxiliary lane. The densities of the first and second type cars stay constant. Cars are placed on road, on auxiliary lane and on on-ramp randomly and with random speed as an initial condition.

3.2. Computational details

Data about the vehicle position for each time step is recorded by the two dimensional array. Every vehicle has a number and their parameters can be found by struct function so it is easy to give behavioral characteristics to vehicle drivers. The update rules defined below are applied from the left hand side end of the road to right hand side of the beginning to each car. Vehicle motion with respect to its position,

velocity, the distance between the preceding vehicle are analyzed in functions out of the main function. So we can easily both insert and remove different arrangements of the program. Vehicle knows the preceding vehicle's number, position, velocity. Distance between the preceding, trailing vehicle's number, position, velocity, distance between the trailing are also known. We can follow each vehicle's motion. Vehicle can look for the preceding and trailing vehicles on the right or left lane for lane changing.

3.3. Local update rules

- **Acceleration:** If acceleration is possible vehicle accelerates by one. The rule is the same with NaSch Model [4].
- **Motion:** If the car cannot change lane, it follows the preceding car right behind it. Aggressive behavior causes more braking and acceleration.
- **Deceleration:** Driver behavior in our model is more aggressive than NaSch model [2]. Drivers go fast as soon as possible and they can stop or decrease their speed more than one level instantly. This behavior model is placed to the code to simulate the general Turkish driver behavior. We investigate the relation between the braking and jam formation.

$$v^n = min(v^n, d^n - 1)d^n = (x^{n+1} - x^n) \tag{3.1}$$

- **Random deceleration:** Vehicle can decelerate one cell with random deceleration probability (P(r)). Deceleration probability is used for driver's unexpected braking one level. The decrease of the car with no reason, based on the flow and jam, contributes to show different driver behavioral characteristic. The source of random deceleration can be watching an accident on the other side of the road while passing, aggressiveness or carelessness of the driver.

$$v(n) = v(n) - 1 \tag{3.2}$$

  with "P(r)"
- **Lane Changing Rules:** If the distance between the car and its preceding car

is less than the speed of the car or if the speed of the preceding car is the same or less than the speed of the car, the car looks for the opportunity to go faster. If the distance to the preceding car on the target lane is larger than the distance to the preceding car on the current lane, or if the position on the target lane (new position after lane changing) is not exactly behind the preceding car on the current lane (which gives the opportunity to go faster in the next iteration), the car accelerates by one. If the prospect position on the target lane is just behind the preceding car, car decelerates for lane changing. This search procedure is valid for both sides and with equal chance the car moves to the left or to the right. This randomization is put into the simulation for showing various drivers' behavior such as not to change lane even it is to their advantage or to change lane even it is to their disadvantage. The deterministic CA Model in single lane highway becomes stochastic in multi-lane highway model which agrees with [13]. Conditions:

$$x^{n+1} - x^n - 1 < v^n \tag{3.3}$$

with

$$v^{n+1} \leq v^n \tag{3.4}$$

$$x_{target}^{n+1} - x^n - 1 > x_{current}^{n+1} - x^n - 1 \tag{3.5}$$

$$(x(n) + V(n))_{target} > x_{current}^{n+1} - 1 \tag{3.6}$$

the car accelerates by one.

$$v_n = v_n + 1 \tag{3.7}$$

$$(x^{n+1} - x^n - 1)_{righttarget} = (x^{n+1} - x^n - 1)_{lefttarget} \tag{3.8}$$

For equal distances between the preceding car on both current lane and target lane,

$$(x(n+1) - x(n) - 1)_{target} = x(n+1)_{current} - 1 \tag{3.9}$$

car changes lane with "lanechangeprob" without acceleration.

- **Extra lane changing rules:** Sometimes, although all cars have max speed around, the car can change lane if there are more unoccupied cells on the right or left lane. The car looks to the right and to the left. If the distance to the preceding car on the current lane is less than the distance to the preceding car on the right or left lane, it moves to the lane which allows for more spacing. This step is not used in the earlier model. If the preceding car on the right lane and the left lane is at the same distance, the car decides to change lanes with 50 % probability. If,

$$v(n)_{neighbours} = v_{max} = 5x^{n+1}_{current} < x^{n+1}_{rightorlefttarget} \tag{3.10}$$

$$x^{n+1}_{current} < x^{n+1}_{rightorlefttarget} \tag{3.11}$$

vehicle shifts to more advantageous lane. For equal distances between the preceding car on both current lane and target lane,

$$x^{n+1}_{righttarget} = x^{n+1}_{lefttarget} \tag{3.12}$$

one lane is preferred with 50 % probability. If vehicle can not change lane due to the parameter that includes vehicle's will to lane changing and the advantage of the lane changing that serves to the driver, it should adapt its speed to the preceding car.

- **Speed adaptation rules:** If the distance between the preceding vehicle is more

than or equal to five cells, vehicle accelerates by one, if the distance is less than five and if vehicle speed is less than the distance, vehicle has to decelerate and it is placed just behind the cell of the preceding car. If vehicle speed is less than the distance although it accelerates by one, it accelerates and decreases the headway in order to move in advance. Headway is kept as small as possible to simulate Turkish driver behavior. Turkish drivers generally prefer to go bumper to bumper. However, vehicle can accelerate only one cell, decelerate more than one if required.

As Turkish drivers do not care much about street signs, therefore we do not place an exit sign, so drivers are not forced to shift right as soon as possible. They can even wait until they come to cell 290 to shift right. This situation is observed as the main cause of the jam formation. The difference from the first model is the deceleration step.

- **Exiting from Off-ramp Rules:** Type 2 cars exit automatically from the off-ramp when they come to the off-ramp. For the cars on the second lane less than 15 cells before the off-ramp, for the cars on the third lane less than 25 cells before the off-ramp, and for the cars on the fourth lane less than 35 cells before the off-ramp, cars begin to move towards right if condition 1 is satisfied, so that they do not miss the off-ramp. The exit sign is not placed on the road. If a type 2 vehicle cannot move to the right with condition 1, then the car looks for the condition 2.
- *Condition 1:* Condition 1 is used for lane changing of vehicles that come to near the off-ramp. The car that wants to move towards right looks at the right lane, if there is a car within 5 cells, we calculate the distance between them.
  - If v+x is less than (the position of the preceding car - 1) at the target lane and v is less than 5, the car increases its speed and moves right just behind the preceding car at the target lane.

$$x_{target}^{n+1} - x^n - 1 > x^n + v^n \tag{3.13}$$

$$v(x) < 5 \tag{3.14}$$

the vehicle accelerates and changes lanes.

$$v^{n'} = v^n + 1 \tag{3.15}$$

– If v+x is more than the position of the preceding car at the target lane (collision), to avoid collision the speed is decreased to the distance between them.

$$x^n + v^n >= x^{n+1}_{target} \tag{3.16}$$

$$v(x) > 1 \tag{3.17}$$

vehicle decelerates according to headway,

$$v^n = v^n - x^{n+1}_{target} - x^n - 1 - 1$$

– If v+x is the same as (the position of the preceding car - 1) at the target lane, car continues with its speed, changes lanes

$$x^n + v^n = x^{n+1}_{target} - 1 \tag{3.18}$$

and its new speed becomes the same with the preceding car's speed.

$$v^n = v^{n+1}_{target} \tag{3.19}$$

– If there is no car at the target lane along five cells , speed is increased by one and continues.

$$x^{n+1}_{target} - x^n - 1 > 5 \tag{3.20}$$

and its velocity is less than (5), vehicle changes lanes and accelerates by one,

$$v^n = v^n + 1 \tag{3.21}$$

- *Condition 2:* For the cars on second lane, 7 cells before the off-ramp,

$$x_{offramp} - x^n - 1 < 7 \tag{3.22}$$

for the cars on third lane, 9 cells before the off-ramp,

$$x_{offramp} - x^n - 1 < 9 \tag{3.23}$$

and for the cars on fourth lane 11 cells before the off-ramp,

$$x_{offramp} - x^n - 1 < 11 \tag{3.24}$$

cars cut the other car on the right lane off for a last chance to exit from the main road. The other car has to stop if necessary at this condition. If there is no car just at the front cell of the car at both the current lane and the target lane, car shifts just to front cell at the right lane. If there is no car at the target lane,

$$v^n < 5 \tag{3.25}$$

vehicle accelerates by one and changes lanes.

$$v^n = v^n + 1 \tag{3.26}$$

- **Auxiliary Lane Rules:** The car on the fourth lane and between cell 335 and cell 360 is forced to shift to right lanes in order to allow the cars on the auxiliary lane enter the road. Otherwise, auxiliary lane is jammed. These cars look both at the preceding and at the trailing car on the target lane. If the car can exit

from auxiliary lane, it looks at the position of the preceding car on the target lane not to collide. If the car will collide with its current speed, it adapts its speed to avoid collision at the entrance.

- **On-ramp Rules:** If the car can enter from on ramp, it looks to the position of the preceding car on the target lane not to collide. If the car can enter the road by accelerating one level, accelerates and enters with accelerated speed. If it will collide with its speed, it adapts its speed to avoid collision at the entrance and it slides to the cell just behind the preceding car on the target lane. After entering, the car adapts speed to the speed of the preceding car. If it cannot enter the main road, it adapts speed to the preceding car. If it comes to the end of the on ramp, it stops and waits for entering. The car number at on ramp is decreased by one and added one car from the beginning of the on ramp if possible.
- **Adding Cars to the Road:** As we are keeping the density of the cars constant, we look at the number of cars leaving the road at the off-ramp and the cars crossing the line of the off-ramp on the main road. We add the sum of these two values to the beginning of the road. This way the concentration between the beginning of the road and the first exit is kept constant. The cars are added to the beginning of the road with the following conditions, true for each lane: If there is a car on any of the first 5 cells on the lane, the new car is placed just behind the first car on the road, and new car's speed becomes the same with the first car on that lane. If there are no cars on the first 5 cells on the lane, we look at the total number of cars exiting (by how much the density decreases)
  - If the total number of cars exiting is 5 or less than 5, the car can be randomly placed on any of the first five cells.
  - If the total number of cars exiting is between 5 and 7, the car can be randomly placed on cells 2-5.
  - If the total number of cars exiting is between 7 and 10, the car can be randomly placed on cells 3-5.
  - If the total number of cars exiting is between 10 and 15, the car can be randomly placed on cells 4-5.
  - If the total number of cars exiting is larger than 15, the car must be randomly placed on cell 5.

4. SIMULATIONS

4.1. Homogeneous Closed (Periodic) Boundary Model

Definition Homogeneity means that there is neither entrance to the road and nor exit from the road. In Kerner's theory, bottleneck term is used for the decrease in the number of lanes on the road such as on-ramp, off-ramp or auxiliary lane [10]. In this version, closed boundary condition is used, it means that there is no change in density. The number of vehicles which leave the road, enter the road from the beginning. Closed boundary condition is generally used for finding theoretical correspondence of vehicle on traffic in hydrodynamic, granular, and gas systems.

In general, lane numbers are respectively from (5) to (0) for space-time and flow-density diagrams. Lane (5) defines auxiliary lane and lane no (0) defines first lane, the lane that vehicles enter to the main road and exit from the main road.

When looking at Figure 4.1, position-time graphics for homogeneous road, there is no jam formation for 30 % density road. However, when density increases, number of vehicles that decelerate for no apparent reason and changes lane aggressively increase and it causes to jam formation as shown in Figure 4.3. The fundamental diagram for 30 % density, shown in Figure 4.2, the flow rate is less than 50 % density in Figure 4.4. The synchronized flow phase and jam phase formation is obviously seen for 50 % density in Figure 4.4

4.2. Heterogeneous Model

Definition Heterogeneous road defines the main road with entrance, and bottleneck. The road explicitly shows different flow behavior due to the fluctuations of density and vehicles have a different attitude about entrance to the road, and exiting from off-ramp.

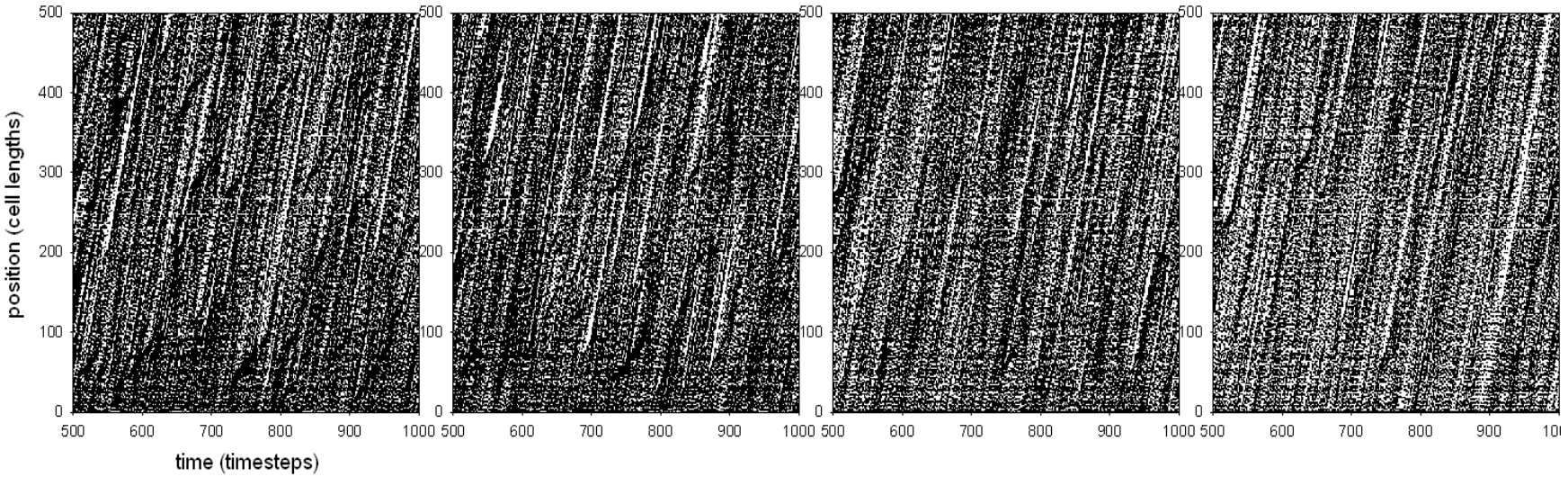


Figure 4.1. Homogeneous road space-time diagram, d(main road): 30 %, d(type2): 5 %, P(r): 10 %, P(lc): 100 %.

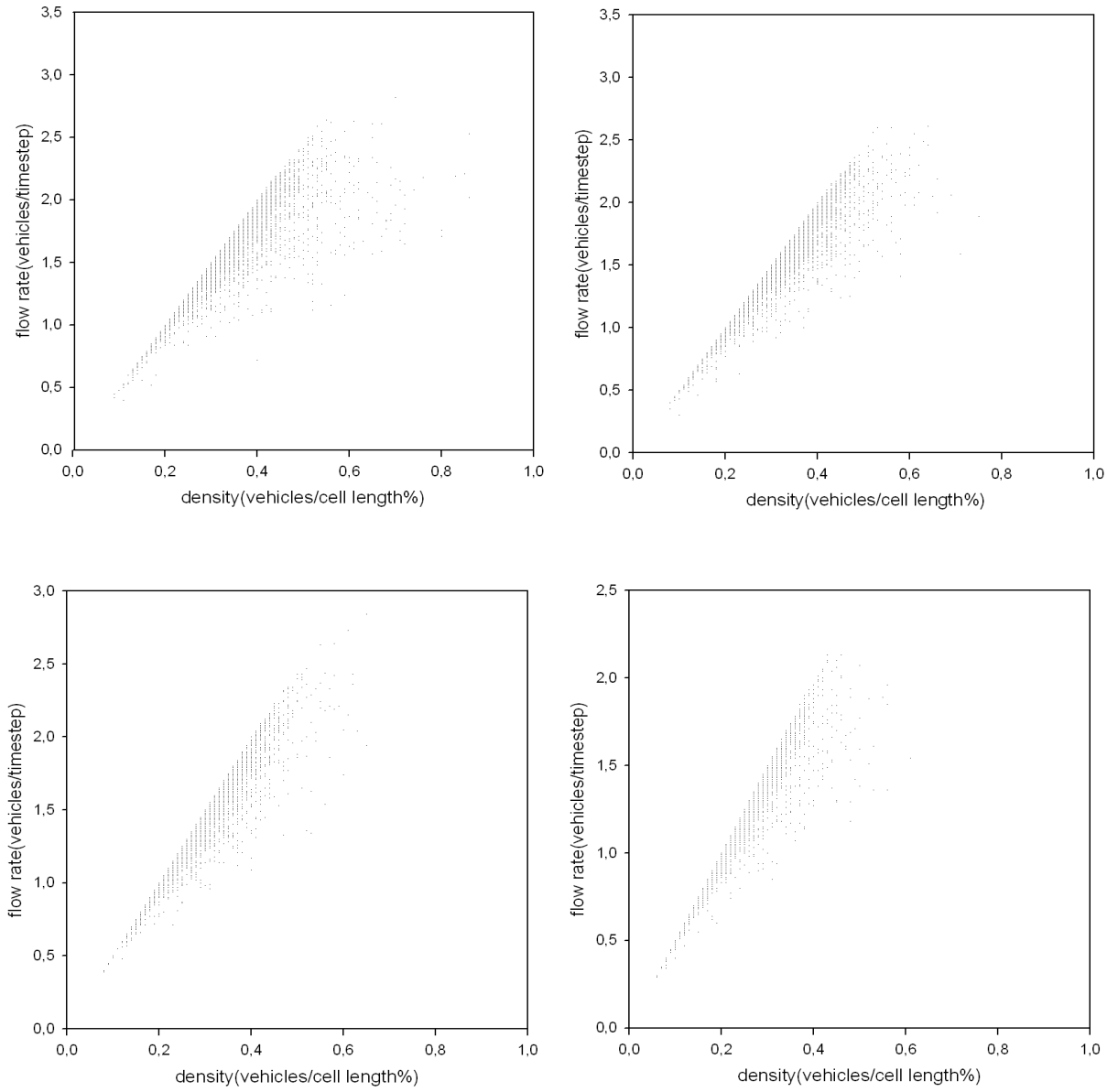


Figure 4.2. Homogeneous Road Fundamental Diagram, d(main road): 30 %,d(type2): 5 %,P(r): 10 %,P(lc): 100 %

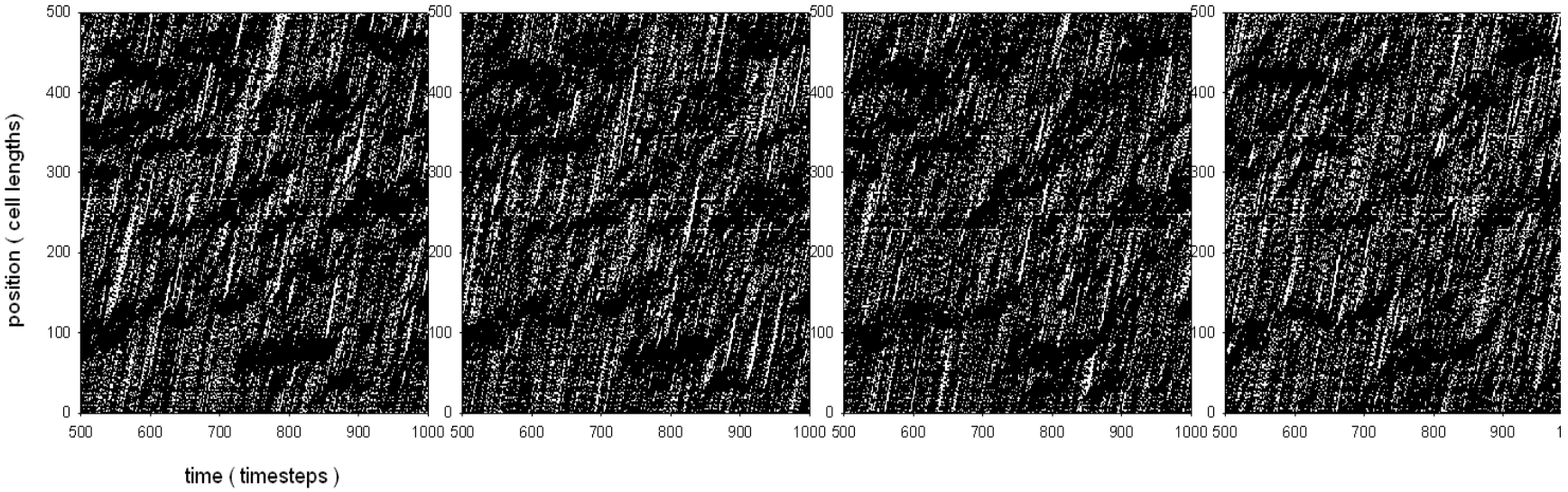


Figure 4.3. Homogeneous Road space-time diagrams, d(main road): 50 %, d(type2): 8.3 %, P(r): 10 %, P(lc): 100 %.

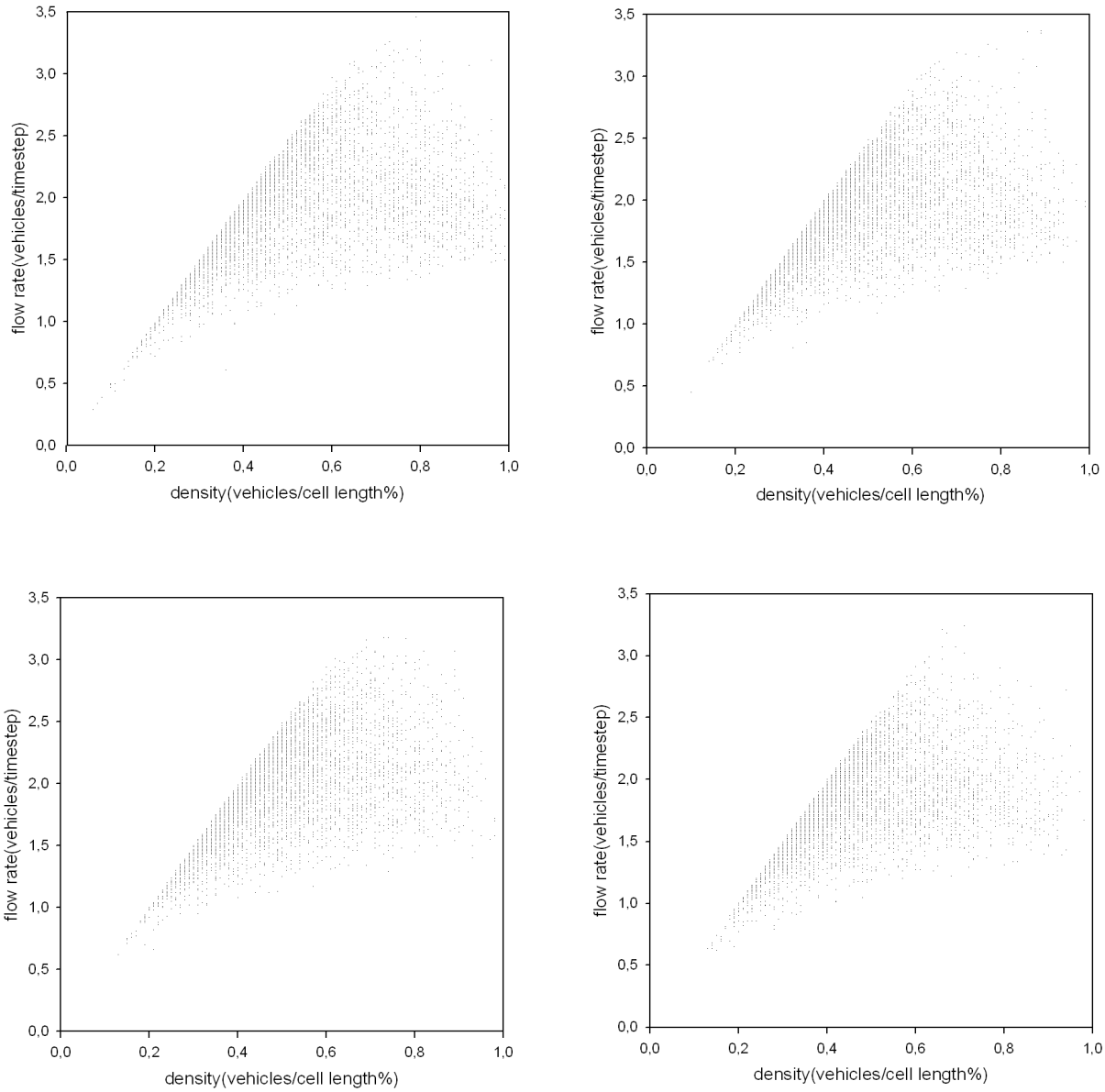


Figure 4.4. Homogeneous Road Fundamental Diagram, d(main road):50 %, d(type2) : 8.3 %, P(r):10 %, P(lc):100 %.

4.3. FSM-Kavacık TEM highway applications

Definition This version of the model is completely simulation of Fatih Sultan Mehmet Bridge Kavacık highway. The road portion that will be simulated and off-ramp, on-ramp and auxiliary lane positions are simulated and scaled microscopically to approximately to 3750 meters. The road portion is shown at Figure 4.5.

4.4. The effect of auxiliary lane entrance to the traffic flow

In our country, the transportation authority makes an arrangement in order to decrease the traffic jam, by taking one lane from the direction from eastern side to western side and adds this lane to the crowded side as an auxiliary lane at the rush hours. We investigate the effect of this auxiliary lane and its position on the traffic flow. The general FSM Bridge-TEM highway model is shown in Figures 4.6 and 4.8 for 30 % and 50 % densities, respectively. The exclusion of auxiliary lane indicates us the effectiveness of auxiliary lane. If there was no auxiliary lane, the jam formation would increase even for 30 % density. FSM Bridge-FSM highway model without auxiliary lane is modeled in Figures 4.7 and 4.9 for 30 % and 50 % densities, respectively. An increase in density increases the jam formation. An increase in jam density is considered as a natural result of an decrease in the number of lanes and also cells.

4.4.1. Effect of random driver deceleration probability

Simple and logical rules seem to have a better result in terms of the traffic flow, whereas driver's disrespect for common traffic rules can result in serious traffic jams. Random slowing down in our model is basically due to frequent and abrupt lane changing behavior commonly seen in our traffic. Only random deceleration probability is changed under the same condition, and the flow looks different. In Figures 4.10 and 4.13, the random deceleration probability is 10 %, but in Figures 4.11 and 4.15, random deceleration probability increases to 50 %. The difference between these cases shows us that this random slowing can have adverse effects on all of the freeway lanes for 30

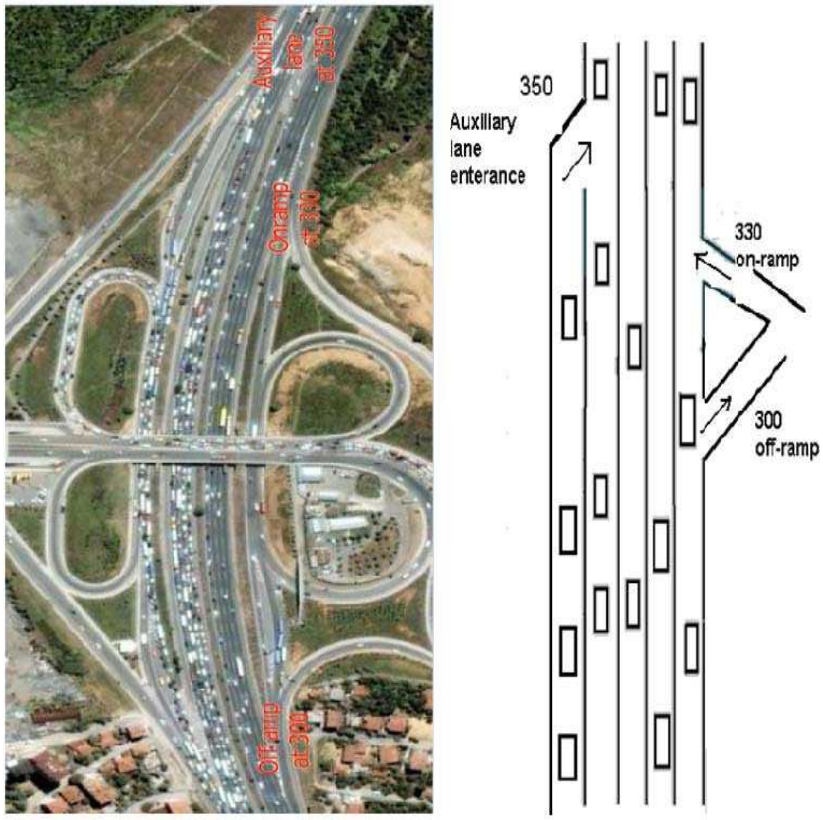


Figure 4.5. The picture of FSM Bridge-TEM Highway

% density road. The jam becomes the general pattern of the road and position-time graphics of this application gives the critical behavior and slope formation. The slope gives us the velocity of jam formation. The other different observation can be seen for 50 % density road with 50 % random deceleration probability. Although the density is 50 %, there is no jam formation around bottleneck in Figure 4.15. Jam expands to all parts of the road. When looking at density-flow diagrams, shown in Figure 4.12 for 30 % and 50 % random deceleration probability, the number of vehicle per lane is not homogeneous like previous model. The flow rate for each lane is different. While flow rate of first and second lane is high, flow rate for auxiliary lane and fourth lane is relatively low. The different characteristic can be observed in Figures 4.14 and 4.16, for 10 % random deceleration probability while jam forms for higher densities, jam is not formed and density is not high as in Figure 4.16 and this result is consistent with space-time diagram, shown Figure 4.15.

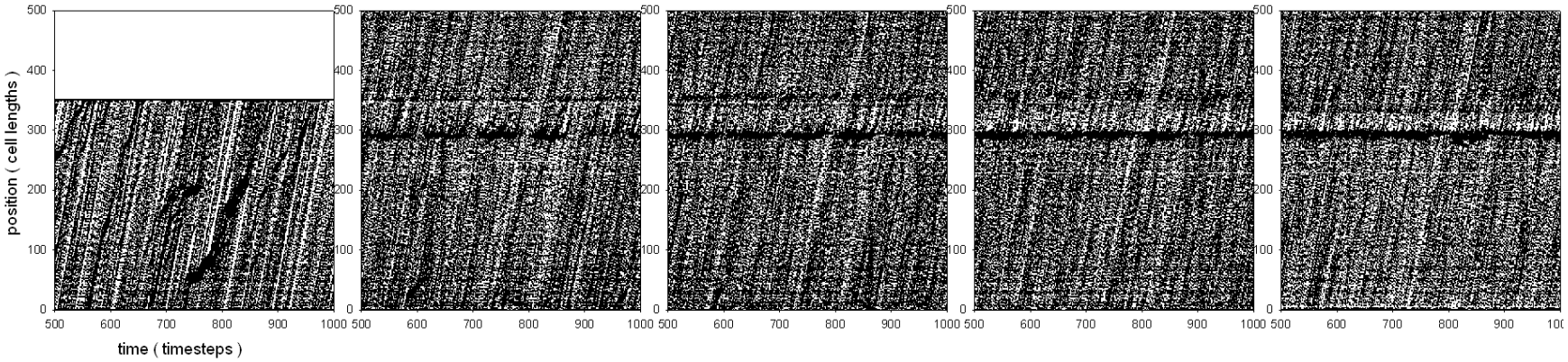


Figure 4.6. FSM Bridge-TEM Highway space time diagram, d(main road):30 %, d(type2): 5 %, P(r): 10 %, P(lc): 100 %, d(onramp): 30 %.

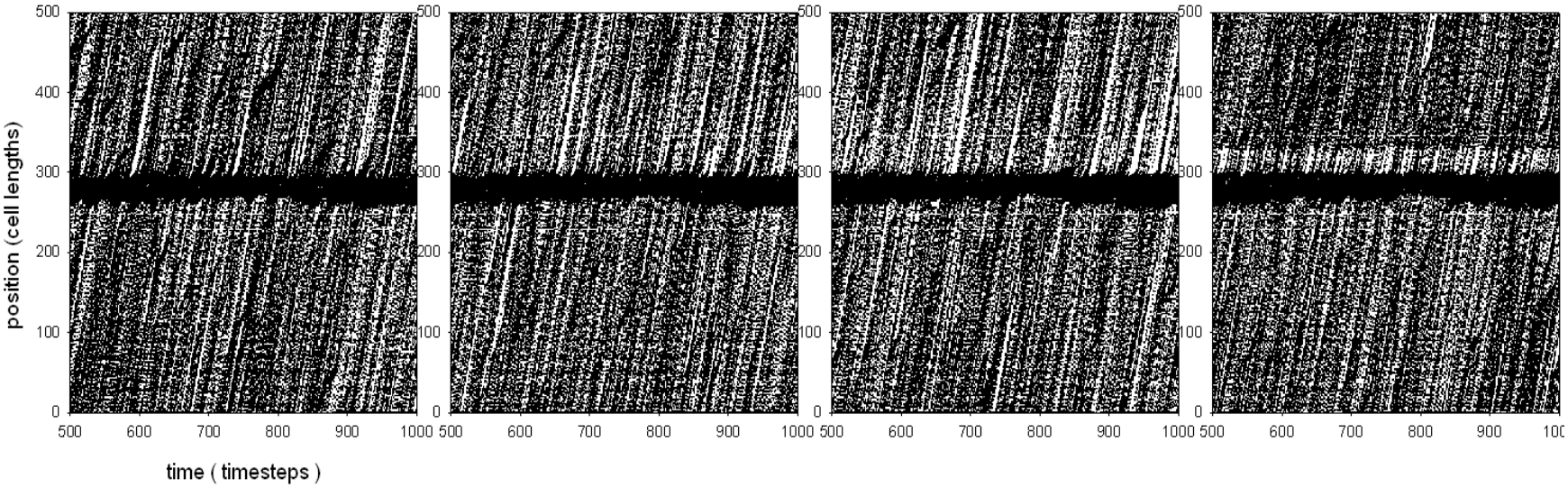


Figure 4.7. FSM Kavacık-TEM highway without auxiliary lane space-time diagram, d(main road): 30 %, d(type2): 5 %, P(r): 10 %, P(lc): 100 %, d(onramp): 30 %.

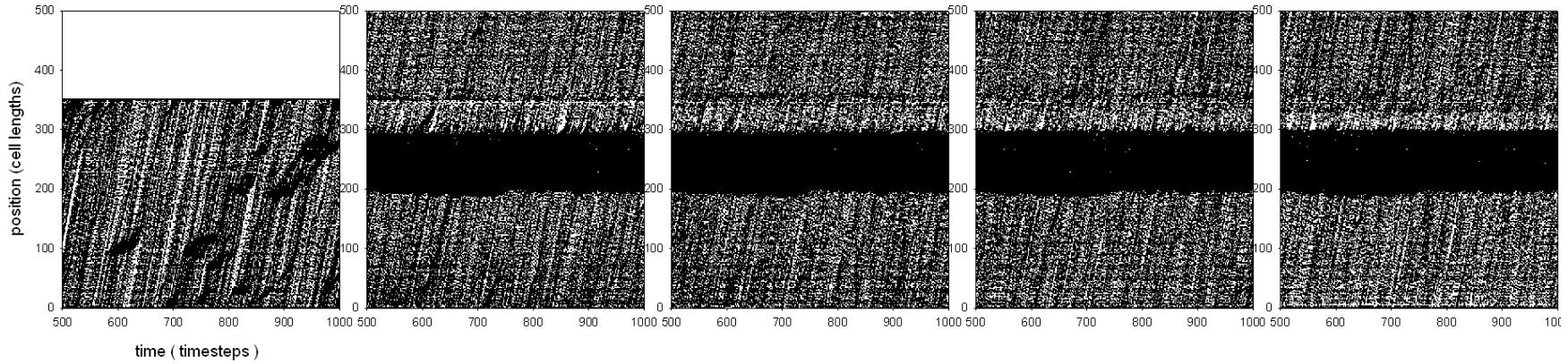


Figure 4.8. FSM Bridge-TEM Highway space-time diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 10 %, P(lc): 100 %, d(onramp): 30 %.

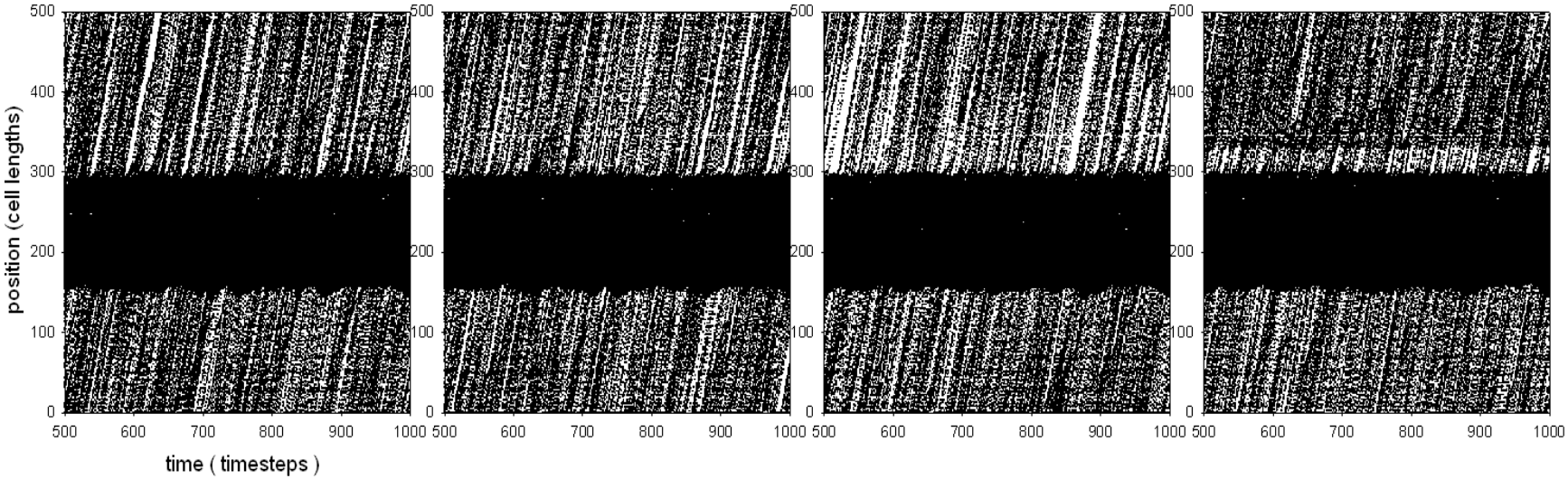


Figure 4.9. FSM Kavacık-TEM highway without auxiliary lane space-time diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 10 %, P(lc): 100 %, d(onramp): 30 %.

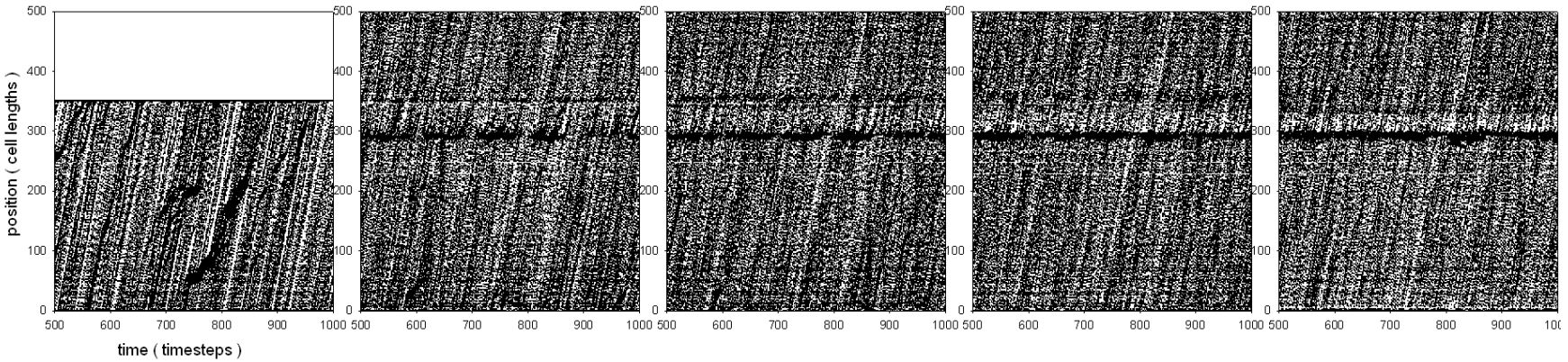


Figure 4.10. FSM Bridge-TEM Highway space time diagram, d(main road): 30 %, d(type2): 5 %, P(r): 10 %, P(lc): 100 %, d(onramp): 30 %.

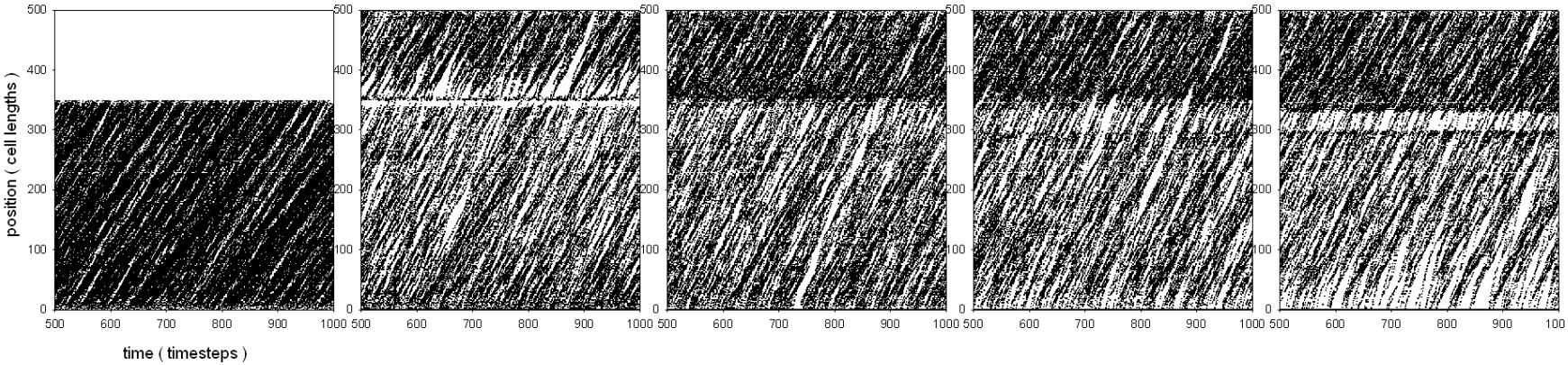


Figure 4.11. FSM Bridge-TEM Highway space-time diagram, d(main road): 30 %, d(type2): 5 %, P(r): 50 %, P(lc): 100 %, d(onramp): 30 %.

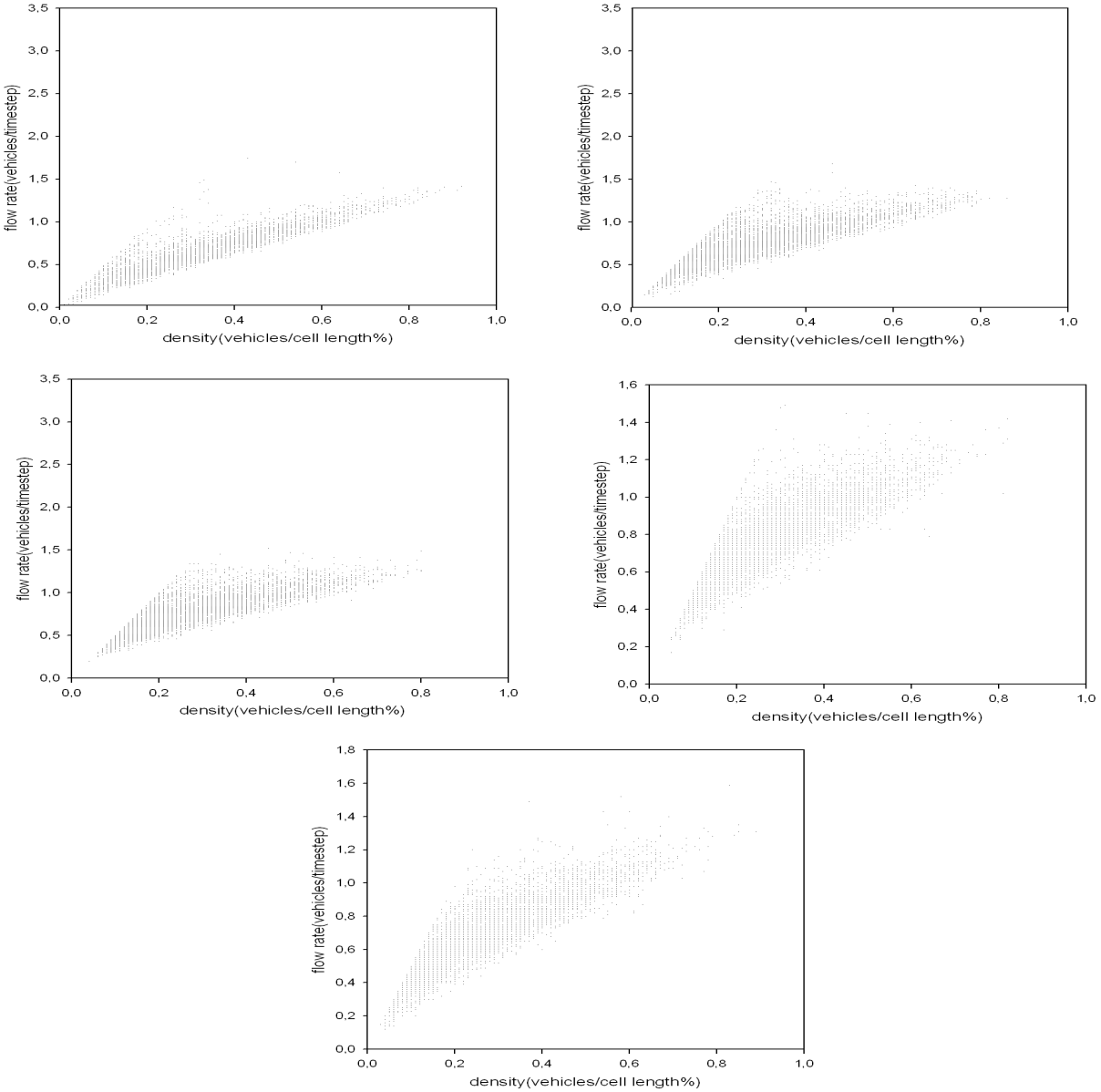


Figure 4.12. FSM Bridge-TEM Highway fundamental diagram, d(main road): 30 %, d(type2): 5 %, P(r): 50 %, P(lc): 100 %, d(onramp): 30 %.

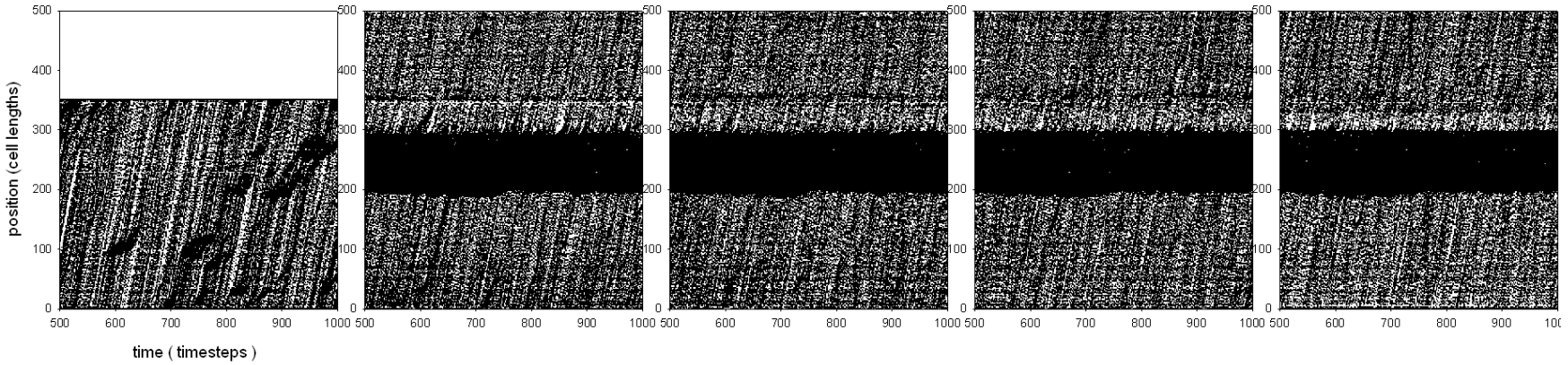


Figure 4.13. FSM Bridge-TEM Highway space-time diagram, d(main road):50 %, d(type2):8.3 %, P(r):10 %, P(lc):100 %,d(onramp):30 %

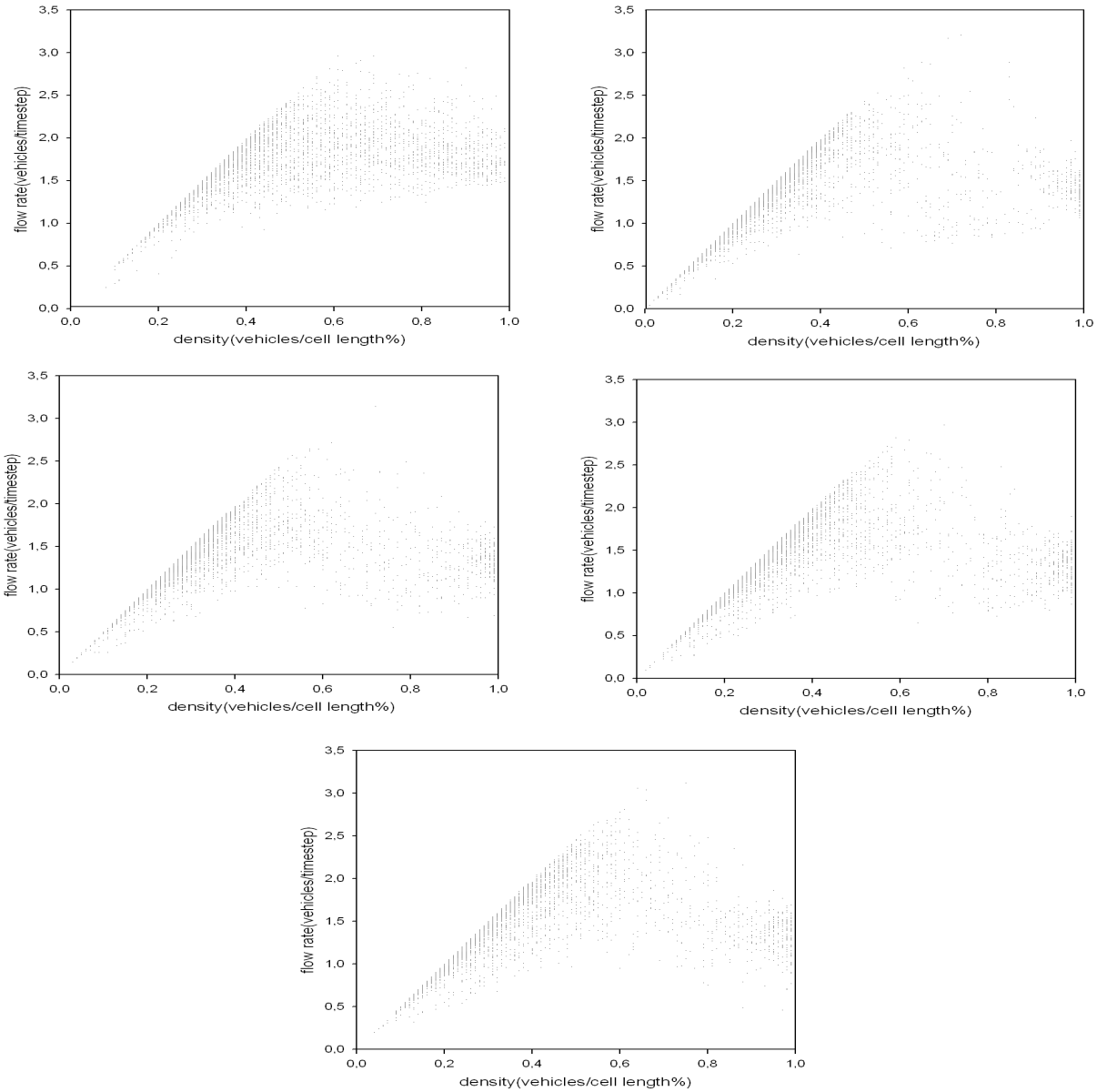


Figure 4.14. FSM Bridge-TEM Highway Fundamental diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 10 %, P(lc):100 %, d(onramp): 30 %.

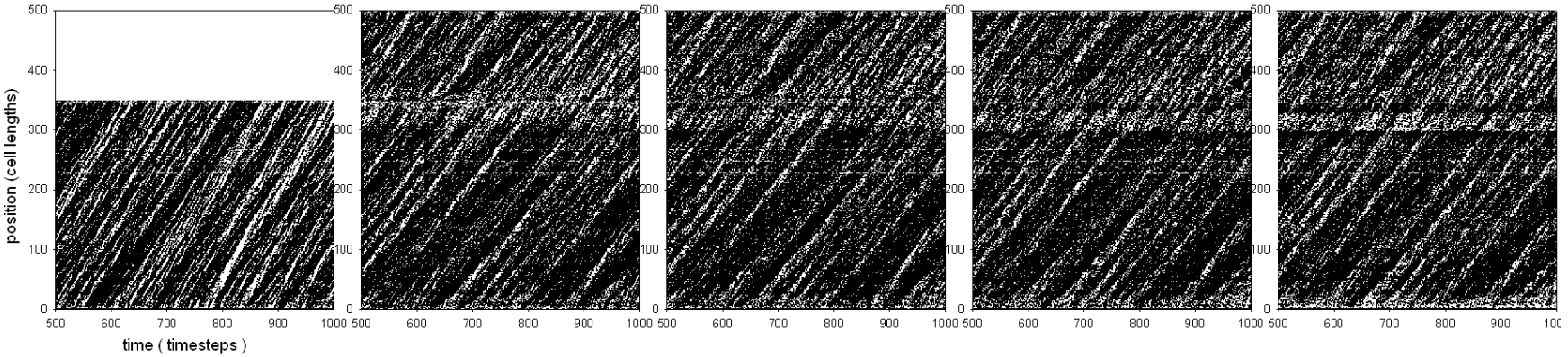


Figure 4.15. FSM Bridge-TEM Highway space-time diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 50 %, P(lc): 100 %, d(onramp): 30 %.

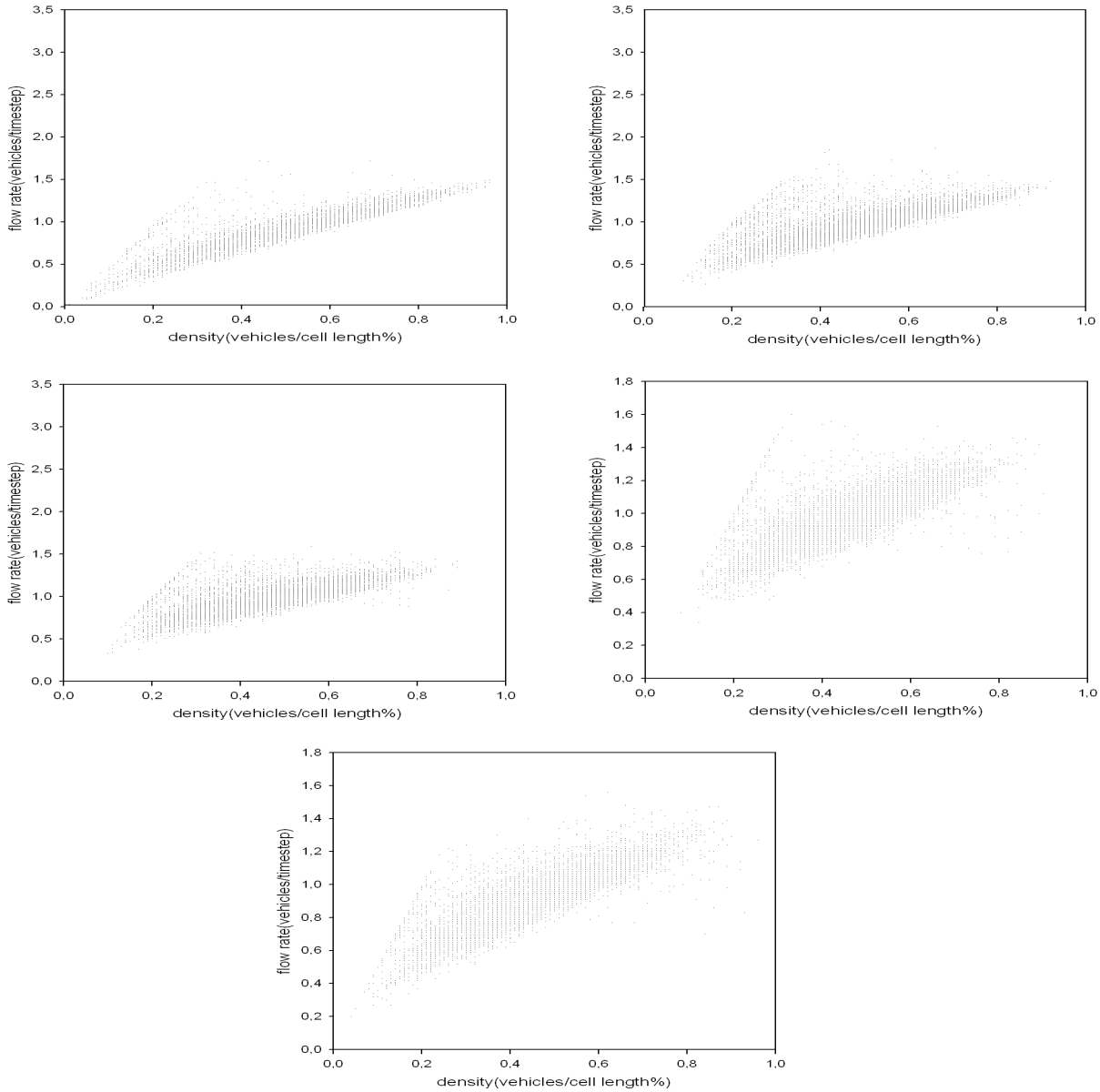


Figure 4.16. FSM Bridge-TEM Highway Fundamental diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 50 %, P(lc): 100 %, d(onramp): 30 %.

4.4.2. The effect of on-ramp density to traffic flow

The important parameter that should be investigated is the fluctuation in the main road density due to entrance and exit lane. The increase in the number of vehicles per kilometer, caused by bottlenecks is obviously proportional to the increase in the density of on-ramp. Figure 4.17 is the space-time diagram for 30 % mainroad density and 50 % on-ramp density and there is an increase in the jam width, but it is small that can be neglected. This situation is valid for Figure 4.18. The increase in on-ramp density does not effects the flow and jam formation.

4.4.3. Addition of behavioral effects to the model

Lane changing and extra lane changing rules as defined above, are valid for all drivers with equal probability. Therefore, a vehicle which has an opportunity to change lanes has to change lanes. Only the guidance system for second type vehicles which will exit from off- ramp, is active in order to shift right. Different lane changing behavior is added to our simulation with Advantage will update rule. Advantage will update rule is peculiar to our model and it is applicable to include different characteristics in different complex systems such as preference model in economy. The selection and giving different alternatives to the driver according to their will and advantage, defines stochastic systems explicitly. In this version of our simulation, we add personal willingness to lane change and advantage of lane changing that serves to driver to go faster or to go ahead simultaneously. In every time step, driver's willingness is defined with a random number. Willingness is absolutely independent of advantage of the action. Deciding whether driver changes lane or not, depends on many parameters. The advantage level is defined with generation of random number with respect to advantage degree, and the decision depends on multitude of advantage and willingness number. Possible number is defined in the Table 4.1 and according to this chart, vehicle changes lanes or not.

Vehicles looks at the preceding car on both left and right lane, if there is no car along 5 cells, prefers to shift the lane where it can go faster.

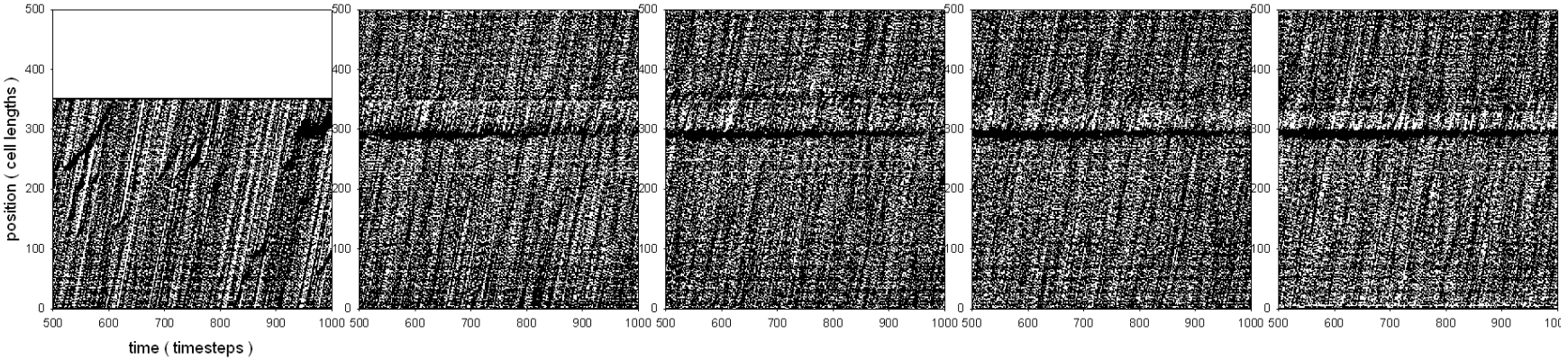


Figure 4.17. FSM Bridge-TEM Highway space time diagram, d(main road): 30 %, d(type2): 5 %, P(r):10 %, P(lc): 100 %, d(onramp): 50 %.

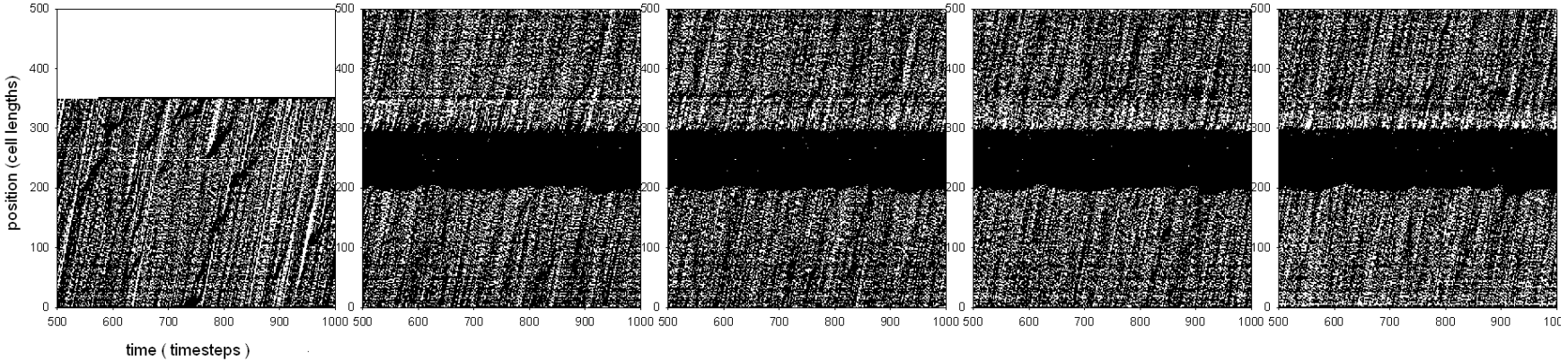


Figure 4.18. FSM Bridge-TEM Highway space time diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 10 %, P(lc): 100 %, d(onramp): 50 %.

- If there is no car along more than 5 cells, the situation is labeled as very advantageous and probability is generated between 0.80 and 1.00.
- If there is a car along 5 cells, the distance is defined for each side. If the velocity of the vehicle and the preceding is maximum (5), and also distance between the preceding on the current lane is smaller than the preceding on the target lane, situation advantage is generated between 0.60 and 0.80 as advantageous. If front car on the right or left lane is farther than the front car on the current lane, it means that vehicle has to decelerate if it stays on its current lane. So probability is generated between 0.60 and 0.80 and situation is labeled as advantageous.

$$H_{plusright} > H_{current} \text{ and } H_{plusleft} > H_{current} \tag{4.1}$$

  Probability between 0.60 and 0.80 is advantageous.
- If headway on the other lanes is also bigger than 5, probability is generated between 0.80 and 1.00 and situation is labeled as very advantageous.

$$H_{plusright} \text{ or } H_{plusleft} \geq 5 \tag{4.2}$$

  Probability between 0.80 and 1.00 is very advantageous.
- If distance between the front and preceding vehicle speed is lower than vehicle speed,So probability is generated between 0.80 and 1.00 and situation is labeled as very advantageous.

$$H_{current} < v \text{ velocity becomes } v_{front} \leq v \tag{4.3}$$

  Probability between 0.80 and 1.00 is very advantageous.
- If preceding vehicle speed is higher than the vehicle speed

$$H_{current} < vandv_{front} > v \tag{4.4}$$

  Probability between 0.60 and 0.80 is advantageous.
- If distance between the front and preceding vehicle speed is higher than or equals

to vehicle speed.

$$H_{current} \geq v \tag{4.5}$$

Probability between 0.60 and 0.80 is advantageous.

- $$H_{plusright} \text{ or } H_{plusleft} < H_{current} \tag{4.6}$$

  Probability between 0.00 and 0.20 is very disadvantageous.
- If the distance between the preceding on left lane and the current lane is the same.

$$H_{plusleft} = H_{current} \tag{4.7}$$

  - Front vehicle speed is higher than the speed of vehicle on current lane front.

$$v_{leftfront} > v_{front} \tag{4.8}$$

    * Front vehicle speed on left is higher than the speed of vehicle on current lane.

$$v_{leftfront} > v \tag{4.9}$$

      Probability between 0.60 and 0.80 is advantageous.
    * Front vehicle speed on left is lower than the speed of vehicle on current lane.

$$v_{leftfront} < v \tag{4.10}$$

      Probability between 0.20 and 0.40 is disadvantageous.

* Front vehicle speed on left equals to the speed of vehicle on current lane.

$$v_{leftfront} = v \tag{4.11}$$

– Front vehicle speed is lower than the speed of vehicle on current lane front.

$$v_{leftfront} < v_{front} \tag{4.12}$$

Probability between 0.00 and 0.20 is very disadvantageous.

– Front vehicle speed equals to the speed of vehicle on current lane front.

$$v_{leftfront} = v_{front} \tag{4.13}$$

Probability between 0.40 and 0.60, there is no need to change lane.

- If the distance between the preceding on right lane and the current lane is the same.

$$H_{plusright} = H_{current} \tag{4.14}$$

– Front vehicle speed on the right is higher than the speed of vehicle on current lane front.

$$v_{rightfront} > v_{front} \tag{4.15}$$

* Front vehicle speed on right is higher than the speed of vehicle on current lane.

$$v_{rightfront} > v \tag{4.16}$$

Probability between 0.60 and 0.80 is advantageous.

* Front vehicle speed on right is lower than the speed of vehicle on current

lane.

$$v_{rightfront} < v \tag{4.17}$$

Probability between 0.20 and 0.40 is disadvantageous.

– Front vehicle speed on the right is lower than the speed of vehicle on current lane front.

$$v_{rightfront} < v_{front} \tag{4.18}$$

Probability between 0.00 and 0.20 is very disadvantageous.

– Front vehicle speed on the right equals to the speed of vehicle on current lane front.

$$v_{rightfront} = v_{front} \tag{4.19}$$

Probability between 0.40 and 0.60, there is no need to change lane.

- The more cell, vehicle can go ahead, vehicle prefers to shift this side due to its will∗advantage parameter.

$$H_{plusright} > H_{plusleft} \tag{4.20}$$

vehicle shifts right.

$$H_{plusright} < H_{plusleft} \tag{4.21}$$

vehicle shifts left.

- If the position that you can go after lane changing on both side is the same, vehicle prefers a lane at random with random generator method.
- If car is just front, in this position car decreases its velocity to zero and it stops.

No car stops for lane changing and it is not feasible.

$$H_{plusright} < H_{plusleft} \tag{4.22}$$

which side driver wants (for which a random number is generated).

In Figure 4.19, Advantage-will lane changing model is used. For 30 % density, the jam becomes the general pattern of the road.

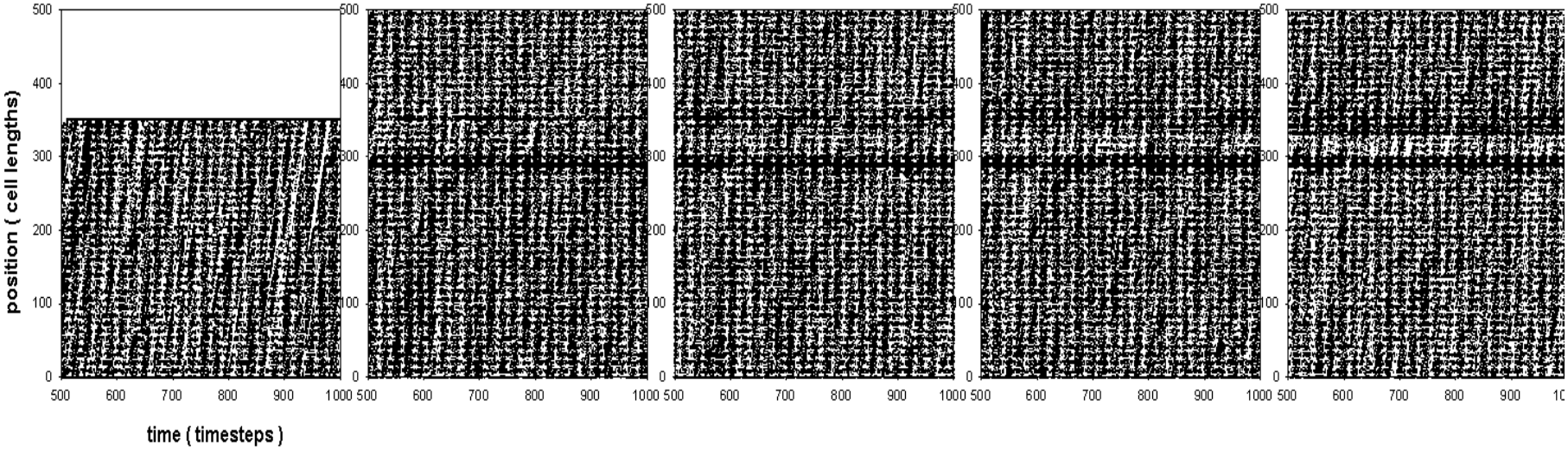


Figure 4.19. FSM Bridge-TEM Highway space time diagram with Advantage Will Model, d(main road): 30 %, d(type2): 5 %, P(r): 10 %, d(onramp): 30 %.

ADVANTAGE WILL MODEL

will	very advantageous	advantageous	no need	disadvantageous	very disadvantageous
wants more(0.80-1.00)	+	+	+	+	0
wants(0.60-0.80)	+	+	+	0	−
to be or not to be(0.40-0.60))	+	+	0	−	−
does not want(0.20-0.40)	+	0	−	−	−
absolutely doesn't want(0.00-0.20)	0	−	−	−	−

Table 4.1. Advantage will model parameter list

5. DATA FROM TRAFFIC CONTROL CENTER IN TURKEY AND ITS COMPARISON WITH SIMULATION

We obtained real traffic data from Traffic Control Center and we analyzed the jam formation and its upstream propagation from velocity-time graphics. RTMS Sensors, placed on the road from FSM Bridge entrance to Küçükbakkalköy were analyzed for a period of one month.RTMS numbers and positions which were used for our analysis are shown in Figure 5.1.

5.1. The RTMS Traffic Measurements

The RTMS (Remote Traffic Microwave Sensor) is a true RADAR (Radio Detection And Ranging) device, designed for traffic sensing applications. It measures the distance to objects in the path of its microwave beam. The ranging capability allows the RTMS to detect stationary and moving vehicles in multiple detection zones. The RTMS microwave beam is approximately 40^o high and 15^o wide. When pointed onto a roadway, the microwave beam projects an oval footprint. Its range is divided into range slices, each 3m. wide. Side wired mounting, the RTMS is located on a roadside pole and is aimed perpendicular to the traffic lanes. Range-slices corresponding to the location of traffic lanes are allocated as detection zones during the setup process. Each detection zone may consist of one or more range slices. A maximum of 8 detection zones can be defined. The length of the detection zone is determined by the width of the beam's footprint.

First, we look at two adjacent RTMS sensors to analyze the velocity of traffic jam front for March 13, 2008.According to generalized traffic theory, traffic jam moves backwards on time on the opposite direction to the traffic flow. So the traffic jam formation on sensor 65 have to reach sensor 93 and the velocity of the upstream or downstream jam front can be obtained by measuring the distance between detectors and the time of the jam formation found by average velocity-time graphics. As you seen

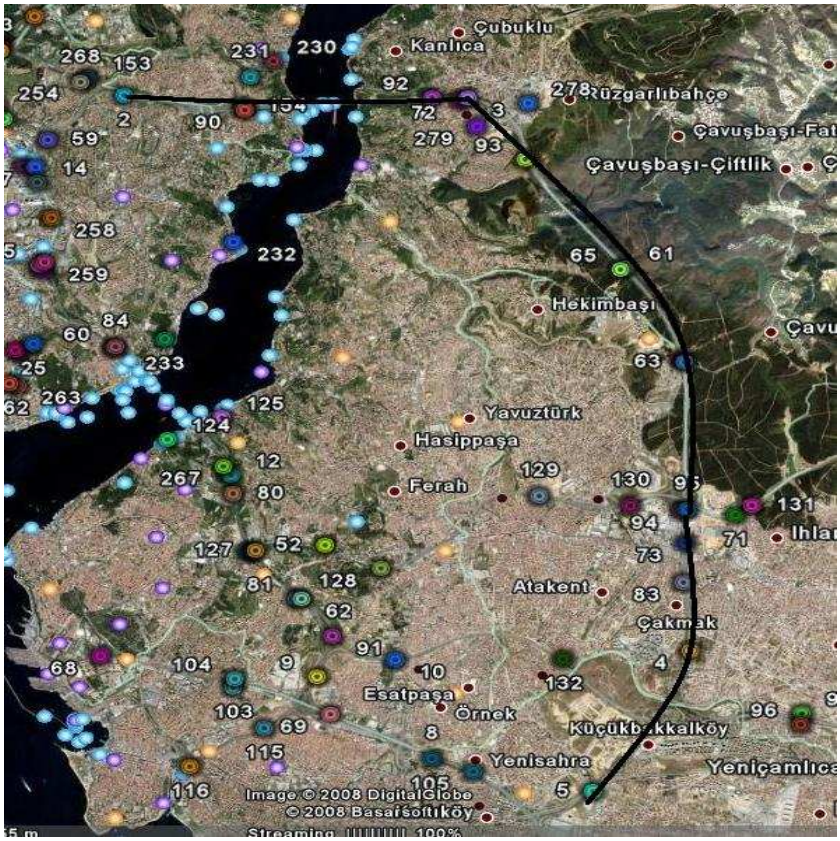


Figure 5.1. RTMS sensor positions on the FSM Bridge TEM highway

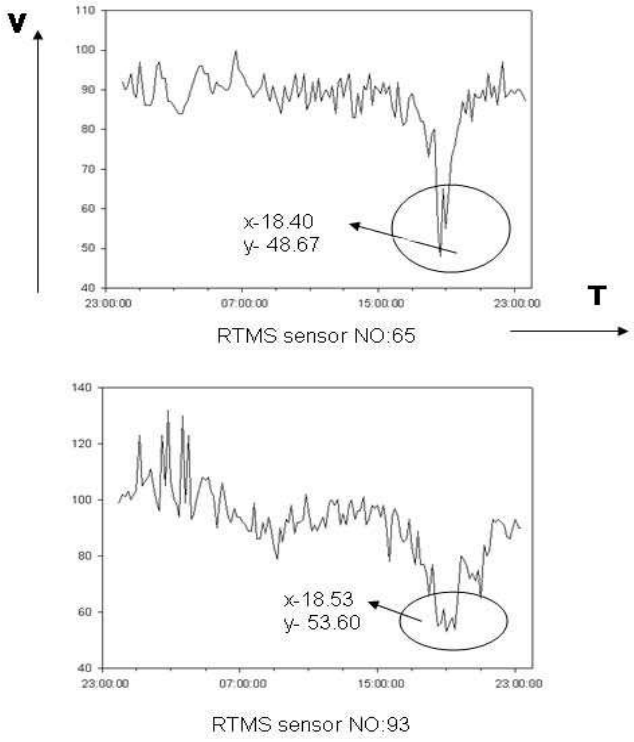


Figure 5.2. FSM Bridge TEM highway RTMS sensor no:65 and 93 velocity-time graphics

on the Figure 5.2, downstream front of traffic jam positions are shown. We measured the distance between sensor 65 and 93. By using these values, the velocity becomes

$$v_{downstream front of jam} = \frac{x_{65} - x_{93}}{t_{65} - t_{93}} \tag{5.1}$$

$$v_{downstream front of jam} = \frac{2350.60}{-13.1000} = -10.84 km/hour \tag{5.2}$$

According to Kerner three phase traffic theory, downstream front velocity of wide moving jam is found to be as -16km/h and Kerner analyzed German traffic control center data, shown in Figure 2.7. He also claimed that this exact velocity value is valid for all road condition, all countries, and for all traffic rules and is independent of all other parameters such as driver behavior. We proved the invalidity of theory for Istanbul highways and approximate velocity values obtained from empirical data could be related to appropriate delay time of drivers with respect to the motion of preceding vehicle. The RTMS sensor numbers on the road portion of our simulation are 93, 279, 3, and 72, are respectively shown in Figure 5.3. The average velocity-time and average velocity-occupancy rate graphics will be shown in Figures 5.4- 5.10 for these sensors.

In Figures 5.9 and 5.10, we analyzed that there is a huge jam formation with respect to Figures 5.5 and 5.7. the space time graphics for sensor 3 shows the jam formation in the morning and in the evening. The jam formation and its motion to the upstream of the flow can be clearly seen in Figures 5.4 and 5.7. In figure 5.9, the flow density values are consistent with theorical fundamental diagram, shown in Figure 2.10 and for lower density,velocity is not high as in Figures 5.5 and 5.8. The Traffic Statistics area displays zone-by-zone traffic data accumulated during the last Message Period. 8 contacts corresponding to the detection zones are provided. Detection status in each range slice is transmitted via “target” messages sent at 100 ms. intervals. RTMS internal firmware uses vehicle detection to accumulate Volume, Occupancy, Average Speed and Classification by length over a user-defined Message Period.

Figure 5.3. Sensor positions of FSM Bridge-Kavack simulation

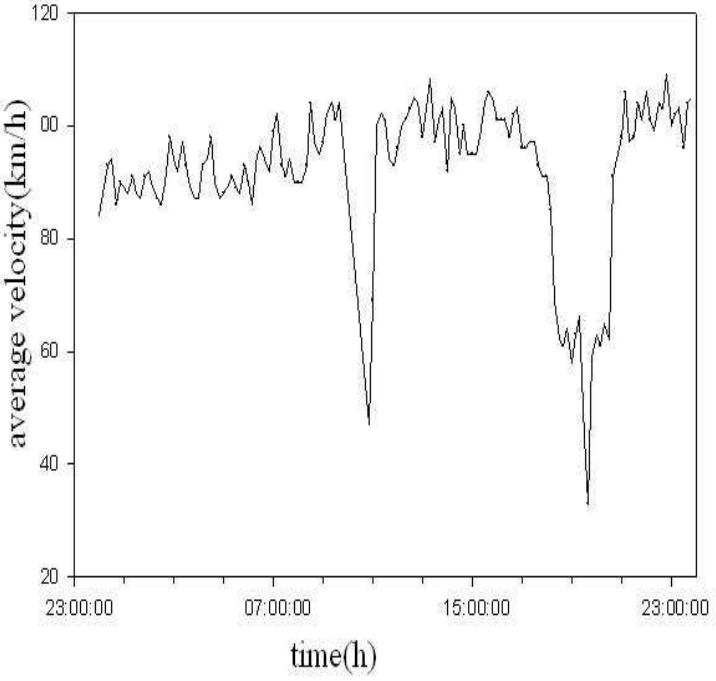


Figure 5.4. FSM Bridge TEM highway RTMS sensor no:93 velocity-time graphics

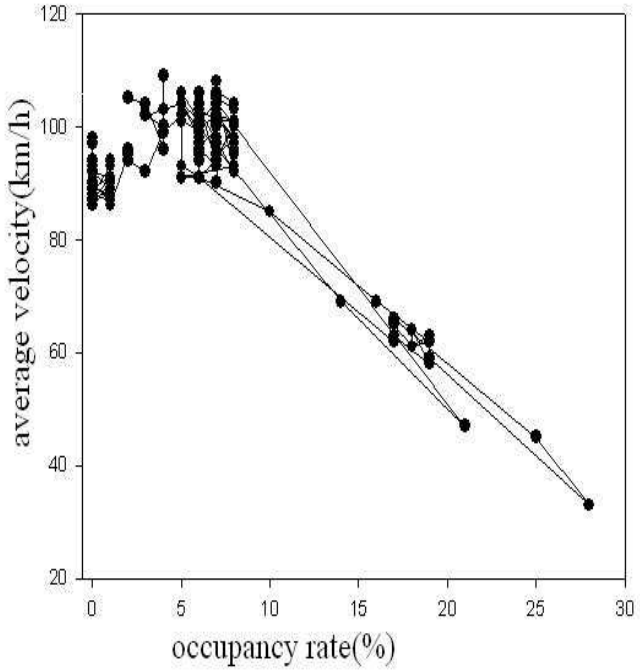


Figure 5.5. FSM Bridge TEM highway RTMS sensor no:93 average velocity-occupancy rate graphics

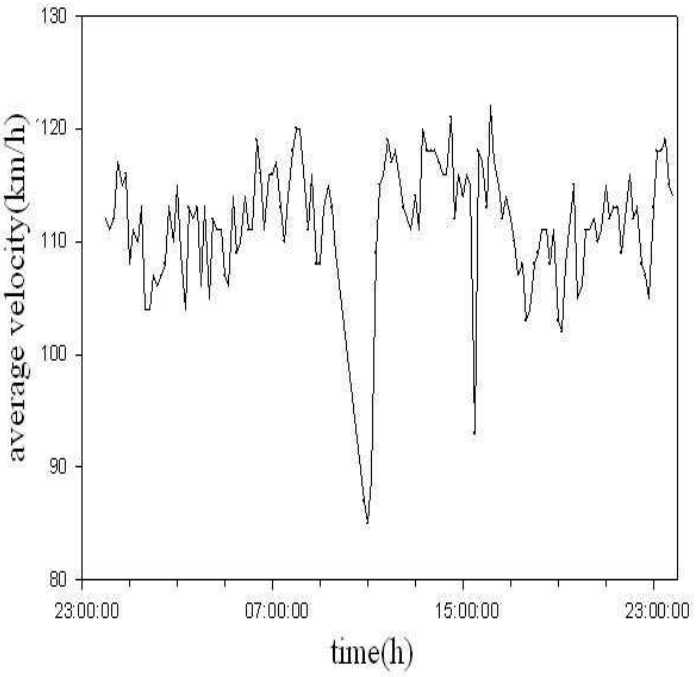


Figure 5.6. FSM Bridge TEM highway RTMS sensor no:279 velocity-time graphics

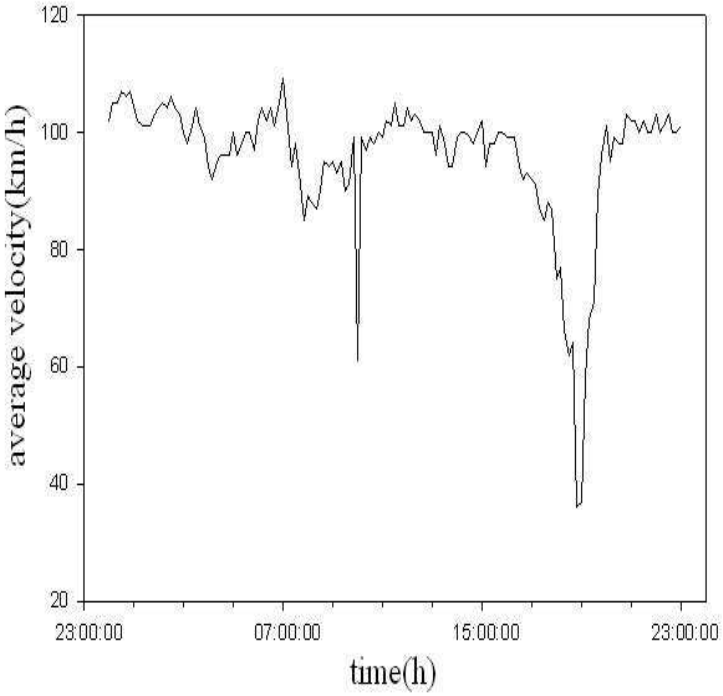


Figure 5.7. FSM Bridge TEM highway RTMS sensor no:72 velocity-time graphics

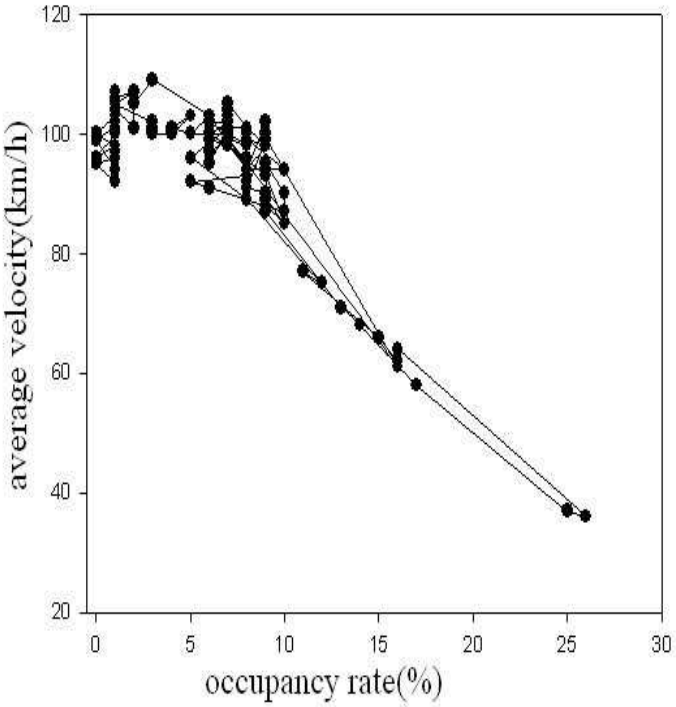


Figure 5.8. FSM Bridge TEM highway RTMS sensor no:72 average velocity-occupancy rate graphics

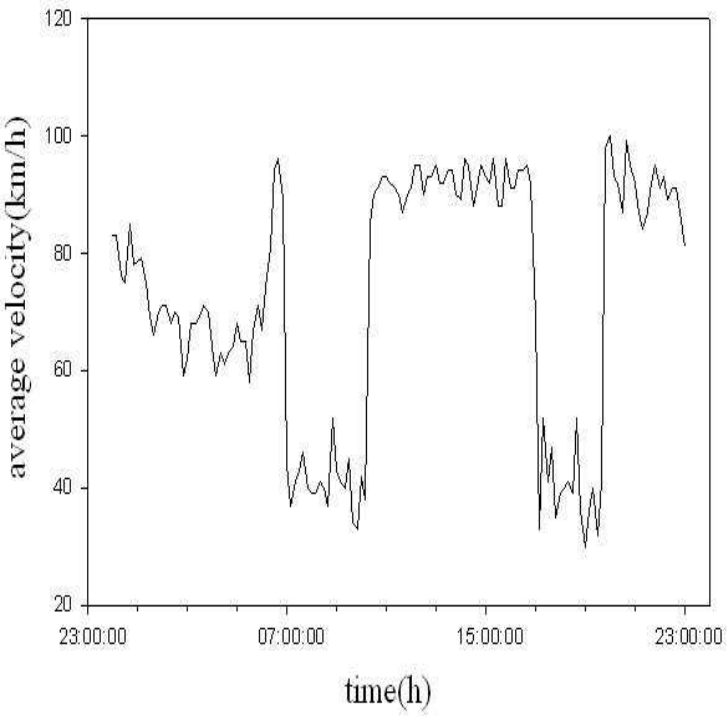


Figure 5.9. FSM Bridge TEM highway RTMS sensor no:3 velocity-time graphics

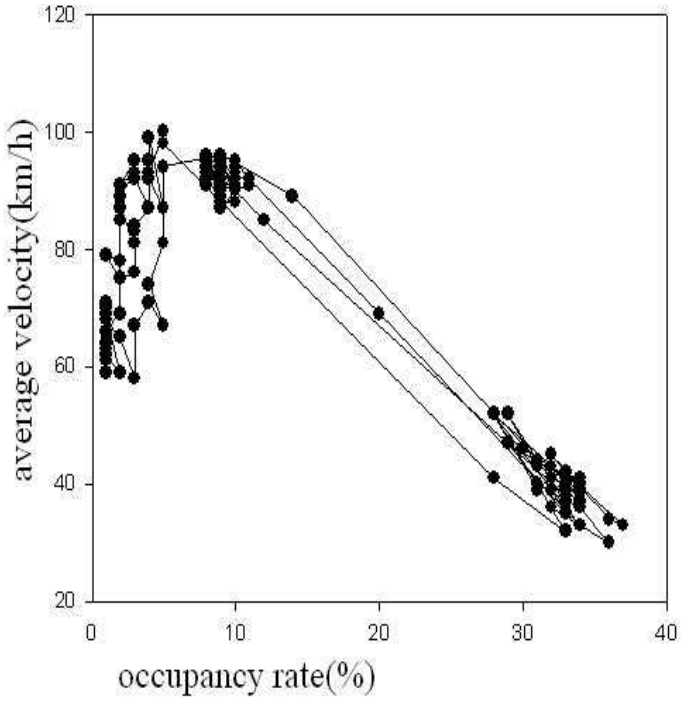


Figure 5.10. FSM Bridge TEM highway RTMS sensor no:3 average velocity-occupancy rate graphics

6. COMPARISON BETWEEN THE TWO CULTURAL TRAFFIC APPROACH

The model described so far is named as MODEL II and the model which was designed earlier by a foreign scholar is named as MODEL I. So Model number is used for making a comparison explicitly. As defined at the reason of the TCA model, the huge jam formation on this area on the evenings appeals us to investigate the effects of the road situation, auxiliary lane injection and its position. In this model, we used open boundary conditions in order to simulate the real traffic clearly. Different aspects and approaches to traffic condition were realized from graphics of simulation data. Exit sign at MODEL I provides the driver with the knowledge of the off-ramp position and helps to decide in advance whether to exit or not. So driver on left side of the road, starts to shift right before approaching the exit. On the other hand, in MODEL II, driver can shift right just before the off-ramp. Drivers can also head car on the right lane off and they cause it to stop not to miss the off-ramp with extra lane changing rules. When vehicle density before off-ramp is increased, due to exit sign vehicles prefer to shift right and this causes jam formation at first lane that exits from the road as in Figure 6.1. In our model, jam formation is spread onto all lanes and it depends on lane changing just before off-ramp as in Figures 4.6 and 4.8.

When we look at the graphics of both simulation, comparing two models designed by a foreign scholar and a turkish one shows the efficiency of the exit sign. In MODEL II, even for 30 % road density,there is a jam formation on the bottleneck(off-ramp) as in Figure 4.6, whereas there is no jam formation on MODEL I 30 % road density as in Figure 6.1. The reason of the jam formation is less organized and more stochastic behavior about shifting right lane to exit from off-ramp in MODEL II. In MODEL II the heavy congestion is caused by not placing the exit sign and not forcing cars to move right in advance. For multi-lane highways with off ramps, we conclude that placing a sign well ahead of the exit is crucial in preventing large scale congestion in that it allows for exiting cars to change lanes without decelerating and for onward

moving cars to move away from the right lane as in Figures 6.1 and 6.2. In MODEL II Figure 4.8, with 50 % road density, there is a more complicated jam formation which is difficult to resolve with respect to MODEL I 50 % road density. The jam formation is spread onto all lanes and congestion differs from MODEL I about jam distribution onto lanes. In MODEL I, there is a congestion on the rightmost lane, since cars try to shift right as they see the exit sign. With logical rules of exiting cars trying to move to the right lane as early as possible, only the traffic on the rightmost lane somehow slows down. However if the drivers wait until the last moment to move towards right, then the rightmost lane is less crowded, but the traffic can come to a standstill on other lanes. In MODEL I, Figure 6.3, no exit sign is placed onto the road, so the jam formation becomes intense. When there is no exit sign in Model 1, for 50 % density road, MODEL I and MODEL II has same condition and their pattern should be similar to each other. Due to the fact that vehicles are not forced to shift right as early as possible, vehicle distribution to lanes should be more homogeneous. The jam formation on the two different model become nearly similar, but the jam formation in MODEL I is not wider than MODEL II. The difference is caused from the fact that MODEL II lane changing rules are more aggressive than first. Although lane changing probability is the same for each model, vehicle lane change advantage behavior is different. The deceleration step is also different from the first model in that vehicle can decelerate more than one level instantaneously. So the jam formation depends on not placing exit sign does not become as huge as in MODEL II.

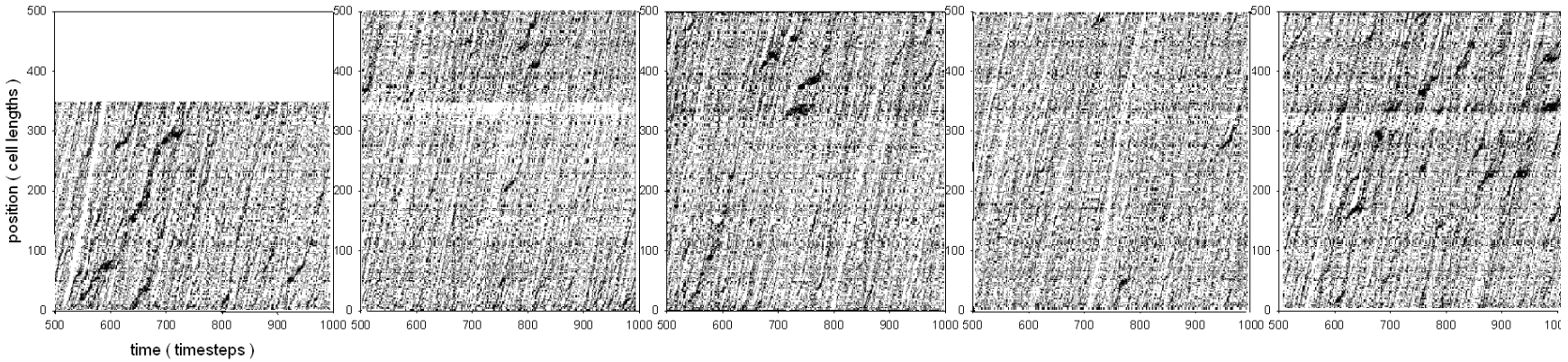


Figure 6.1. Model I FSM Bridge-TEM Highway space-time diagram,d(main road):30 %,d(type2):5 %,P(r):10 %,P(lc):30 %,exit sign: yes,d(onramp):30 %

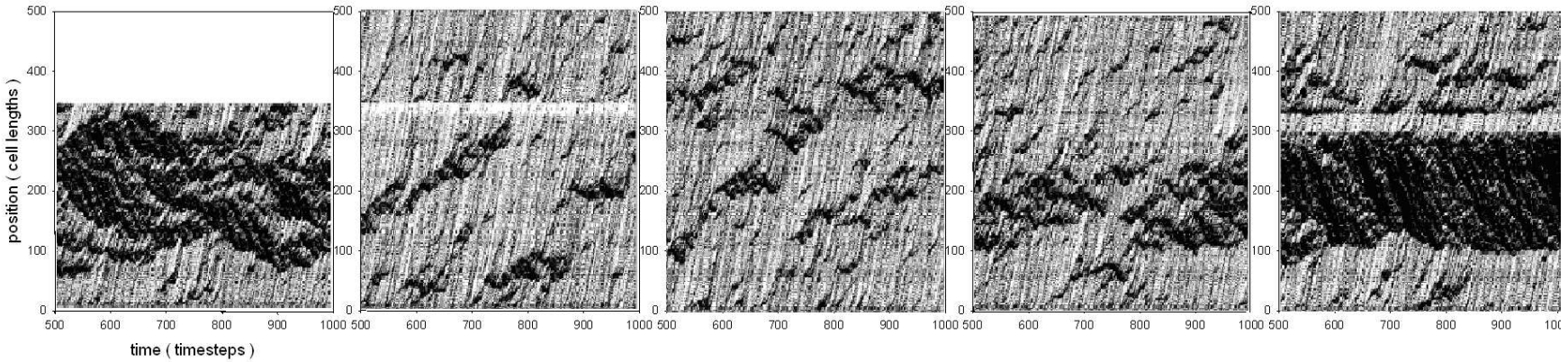


Figure 6.2. Model I FSM Bridge-TEM Highway space-time diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 10 %, P(lc): 30 %, exit sign: yes, d(onramp): 30 %.

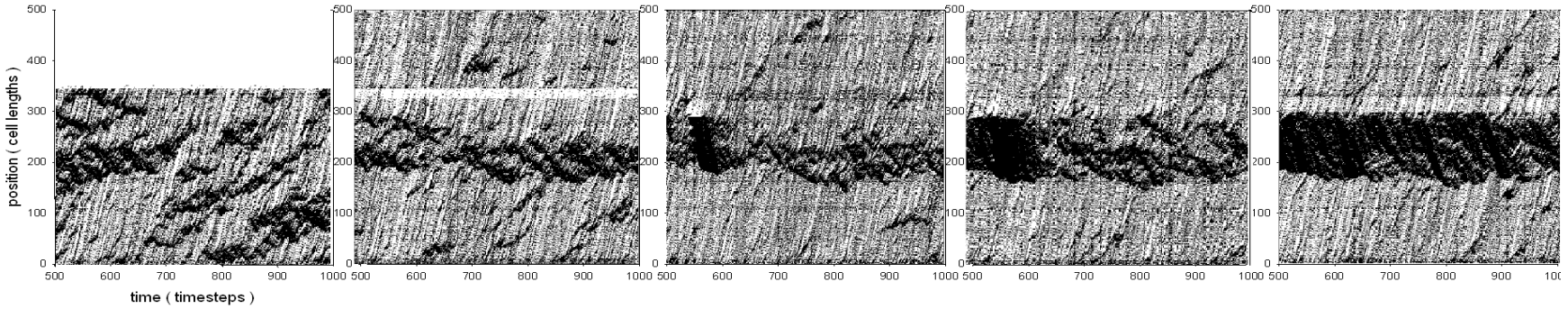


Figure 6.3. Model I FSM Bridge-TEM Highway space-time diagram, d(main road): 50 %, d(type2): 8.3 %, P(r): 30 %, P(lc): 100 %, exit sign: no, d(onramp): 30 %.

7. CONCLUSION

A traffic jam occurs when cars slow down, and the slowing trend continues backwards like a domino effect. As long as there are more cars approaching from behind, the traffic congestion travels in a wave. As the first car stops, the following cars must also stop. Even when the first car begins to move again, additional approaching cars have to stop farther down the road, and so the congested area travels backward in a wave until traffic is light enough for it to dissipate.

It has been found that individual incidents (such as accidents or even a single car braking heavily in a previously smooth flow) may cause ripple effects which then spread out and create a sustained traffic jam when otherwise, normal flow might have continued for some time longer. Jams can arise spontaneously, triggered by minor events (butterfly effects) such as an abrupt steering maneuver by a single motorist.

We have observed two distinct phases in unobstructed flow caused by the formation of spontaneous jams [1]. In figures 4.1 and 4.3, there is no change on the density value and periodic boundary condition is used. We know the capacity of the road from Figure 4.2, and for 30 % density, there is no jam formation on a homogeneous road, but for 50 % density, the spontaneous jam formation can be seen depending on the random deceleration of vehicles. Although Random deceleration probability is the same for both cases, jam formation is seen on 50 % density roads.

Using Nagel-Schrenkenberg model, it is possible to model complex traffic patterns including careless drivers. Especially MODEL I, Figure 6.3, which the exit sign is not placed onto the road and all cases in MODEL 2 simulate the daily traffic conditions experienced by the authors every evening quite accurately [2]. MODEL I is designed by a foreign scholar and MODEL II, this model is designed by a Turkish scholar. The exit sign and less aggressive and more organized behavior for entering from off-ramp (characteristics of the American style of driving) decrease the jam formation explicitly. MODEL II clearly explains the reason of the jam formation on the FSM direction from

the bridge to Kavacık in the evening.

In MODEL II, as Turkish drivers do not care much about street signs, we do not place the exit sign, so drivers are not forced to shift right as soon as possible. They can even wait until they come to cell 290 to shift right. This situation is observed as the main cause of the jam formation. The difference from the first model is the deceleration step. Driver behavior in MODEL II is more aggressive than the first's. Drivers go fast as soon as possible and they can stop or decrease their speed more than one level instantly. This behavior model is placed in to the code to simulate the general Turkish driver behavior. However, in MODEL I Figures 6.1 and 6.2, only first lane and auxiliary lane are congested, in MODEL II all lanes are congested, the reason being more aggressive behavior. In MODEL II cars change lanes with less care and they accelerate and decelerate more. The difference on the speed is the source of the jam formation. The congestion is shared by all lanes. In MODEL I, if the lane changing probability becomes 100 %, the pattern becomes similar to MODEL II. In MODEL II, the congestion after off-ramp should be taken into consideration, the possible cause of the heavy congestion at near the off-ramp is the congestion after off-ramp. The congestion at on-ramp entrance can travel backwards and contribute to the jam formation. The bottleneck at cell 330 (entering on-ramp) causes congestion, according to Kerner's three phase traffic theory [10], the pattern can be the wide moving jam formation at the bottleneck. In both Daganzo [29] and Kerner [10], wide moving jam can not occur in free flow, and the transition from synchronized flow to wide moving jam only can be seen on traffic. The line formation with respect to time around the bottleneck is the empirical feature of the wide moving jam and is named as the breakdown phenomenon as in Figures 4.6 and 4.8. The other important result of distance-time graphics is the curvature on the line around bottleneck due to car accumulation, as shown in Figures 4.11 and 4.15 depends on an increase in the random deceleration probability [3].

The important difference between MODEL I and MODEL II is that at the same density, MODEL II is more congested than MODEL I due to the high braking probability in MODEL II. The comparison between MODEL II figure 4.10 and 4.11, Figure

4.13 , 4.15 shows the effect of the random deceleration probability. Due to the abrupt deceleration (braking) caused the variant gap distribution and the behavior of the gap distribution shows that the traffic can exhibit a self organized criticality for high values of random deceleration probability as found at Fourrate and Loulidi research [30] In MODEL II, when we increase the deceleration probability from 10 % to 50 %, the critical behavior is explicitly seen, lines become less congested and the relation between the self organized criticality defined by Per Bak [28] and this pattern can be investigated in future research. The congestion on the auxiliary lane at MODEL II , based on the lack of entering the main road from auxiliary lane, is more than other four lanes. Once vehicles have passed the location that is the capacity limit, drivers are able to begin resuming their desired speed. There is a gradual acceleration. As the vehicles accelerate, the spacing would increase just to keep the time(headway) constant, since headway is spacing divided by speed. In order for the headway to increase, it can be seen that spacing would have to increase even more.

As shown in Figures 4.17 and 4.18, when the on-ramp density is increased from 30 % to 50 %, there is an increase on the width of the jam formation. The same situation can be observed for the exclusion of auxiliary lane. When we exclude auxiliary lane with the aim of observing the efficiency of the auxiliary lane entrance to the road, the congestion becomes little wider than the road with auxiliary lane. The actual cause of the increase of the jam formation is an increase in the number of vehicles per lane.

APPENDIX A: HIGHWAY SIMULATION CODE

```
#include <stdio.h>
#include "stdafx.h"
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
#define ITER                    1000        // # of iteration
#define CELL                     500         // # of cells-main road
#define MINSPEED                   1        //minimum possible speed of car
#define MAXSPEED                   6        // max speed limit of each car
#define LANE                       6        // # of lanes-main road
#define NUMBER                 11000        // car number
#define INITIALDENSITY            30        //initial density-main road
#define ONRAMPBEGIN              300        //onramp beginning position
#define ONRAMPEND                330        //ONRAMP merging end
#define ONRAMPLANE                 0        //ONRAMP lane number
#define ONRAMPDENSITY             30         //ONRAMP density
#define AUXLANEMERGE             350        // auxlane merging point
#define AUXLANEDENSITY            30        //AUXILIARY ROAD density
#define AUXLANEBEGIN               0        //AUXILIARY ROAD begins
#define AUXLANE                    5        //AUXILIARY LANE number
#define OFFRAMP                  300                //OFFRAMP position
#define TIMEAVERAGE               10        //detector counting time
#define type1dense                25        //density of vehicle that do not exit
#define type2dense                 5         // density of vehicle that exit

int road[CELL][LANE]={0};
short int trace[CELL][LANE]={0};
int carnum=0,type1carnum=0,type2carnum=0;
float dense[100][5] = {0};
float flow[100][5] = {0};
float avespeed[100][5]={0};
int type2carcome=0,type1carcome=0,leftgone=0,rightgone=0;

struct car{
int lane;
int speed;
int pos;
int front;                  //car number of the preceding car
int frontspeed;            // front's speed
int frontdistance;         //how many cells between the car and the front.
int frontpos;               //the front's position
int back;                  //car number of the preceding car
int backspeed;             // front's speed
int backdistance;          // how many cells between the car and the front.
```

```
int backpos;
int nearright;
int nearrightpos;
int nearleftpos;
int nearleft;
int nearleftspeed;
int nearrightspeed;
int type;                    //whether exits from OFFRAMP or go on to the end of the
road.
int exit;
int shiftornot;
int ok;
int entry;

void inspectfront (struct car engine[],int i,int t);
void onrampinspectfront (struct car engine[],int i,int t);
void inspectback(struct car engine[],int i,int t);
void inspectnear(struct car engine[],int i,int t);
void onrampadaptspeed(struct car engine[],int i,int t);
void adaptspeed (struct car engine[],int i,int t);
void headoff(struct car engine[],int i,int t);
void onramp(struct car engine[],int i,int t);
void lanechangecareless(struct car engine[],int i,int t);
void lanechangeadvantage(struct car engine[],int i,int t);
void makecellforaux(struct car engine[],int i);
void  nearofframp(struct car engine[],int i,int t);
void  aux(struct car engine[],int i,int t);
void offrampexit(struct car engine[],int i,int t);
void vacancy(struct car engine[],int i,int t);
};

      int myrandom ( int max )
{   return (rand() * max) / 32767; };

double myrandom ( double max )
{  return (rand() * max) / 32767.0; };

int max( int a,int b) //function finds maximum of  two values.
{      if(a>b)
           return a;
             else
           return b;
};

int min(int a,int b)                    //function finds the minimum of two values.
{
     if(a<b)
        return a;
     else
```

```
        return b;
}

void car::inspectfront (struct car engine[],int i,int t)
{
      int j=0,speedgap=0;
      int c=engine[i].pos;
      int l=engine[i].lane;

for(int x=1;x<=CELL;x++) //looking for the preceding car
 {

if(c+x==CELL && l!=0 && l!=5 || c+x==ONRAMPEND+1 && l==0 ||
c+x==AUXLANEMERGE+1 && l==5 || engine[i].type==2 && c+x==300 && l==1)

{     engine[i].front=0;
      break;
}

 if(road[c+x][l]!=0)
{
        j=road[c+x][l];       //finding which car there is.
        engine[i].frontpos=engine[j].pos;     //its pos is named as frontpos
        engine[i].front=j;
        engine[i].frontspeed=engine[j].speed;   //j.speed is named as frontspeed
        engine[i].frontdistance=x-1;
//wrt to their position,finding the distance between them.
        speedgap=engine[i].frontspeed-engine[i].speed;
        break;
}
}
};

void car::onrampinspectfront (struct car engine[],int i,int t)
{
      int j=0;
      int c=engine[i].pos;
      int l=0;

                for(int x=1;x<=ONRAMPEND;x++) //looking for the preceding car
                 {
                          if(c+x==ONRAMPEND)
                          {engine[i].front=0;
                          break;
                          }
                          if(road[c+x][0]!=0)
                          {
                     j=road[c+x][0];       //finding which car there is.
                     engine[i].frontpos=engine[j].pos;  //its pos is named as frontpos
```

```
                    engine[i].front=j;
                    engine[i].frontspeed=engine[j].speed; //j.speed is named as frontspeed
                   engine[i].frontdistance=x-1;
                //wrt to their position,finding the distance between them.
                   break;
                        }
               }
};

void car::inspectnear(struct car engine[],int i,int t){
      int j=0,k=0;
      int c=engine[i].pos;
      int l=engine[i].lane;

                        if(road[c][l+1]!=0 && l!=4)
                         {
                                j=road[c][l+1];
                            engine[i].nearleft=j;
                                engine[i].nearleftpos=engine[j].pos;
                               engine[i].nearleftspeed=engine[j].speed;

                         }
                        else
                        {
                            engine[i].nearleft=0;
                                engine[i].nearleftpos=0;
                                engine[i].nearleftspeed=0;}

                        if(road[c][l-1]!=0 && l!=1)
                        {
                                k=road[c][l-1];       //finding which car there is.
                                engine[i].nearright=k;
                                engine[i].nearrightpos=engine[k].pos;
                                engine[i].nearrightspeed=engine[k].speed;
                            //j.speed is named as frontspeed

                        }
                        else
                        {engine[i].nearright=0;
                        engine[i].nearrightpos=0;
                        engine[i].nearrightspeed=0;}
};

void car::inspectback(struct car engine[],int i,int t)
{
      int j=0,speedgap=0;
      int c=engine[i].pos;
      int l=engine[i].lane;
```

```
   for(int x=1;x<=CELL;x++) //looking for the preceding car
                {
                        if(c==0)
                        {
                    engine[i].front=0;
                        break;
                        }
                        if(road[c-x][l]!=0)
                        {
                                j=road[c-x][l];       //finding which car there is.
                                engine[i].backpos=engine[j].pos;  //its pos is named as frontpos
                                engine[i].back=j;
                                engine[i].backspeed=engine[j].speed;
                            //j.speed is named as frontspeed
                                engine[i].backdistance=x-1;
                            //wrt to their position,finding the distance between them.
                                speedgap=engine[i].backspeed-engine[i].speed;
                                break;
                        }
                }
};



void car::headoff(struct car engine[],int i,int t)
{
int c=engine[i].pos;
int l=engine[i].lane;

if(road[c+1][l]==0 && road[c+1][l-1]==0)//car looks for the front cell of the target lane
{
      road[c][l]=0;
      road[c+1][l-1]=i;   //new position after lane changing.
   engine[i].pos=c+1;
   //shift right without giving signal and trailing car has to adapt speed wrt to you.
      engine[i].lane=l-1;
      engine[i].speed=engine[i].speed;
      engine[i].ok=1;
      engine[i].shiftornot=1;
}
else if (road[c+1][l]==0 && road[c+2][l]==0 && road[c+2][l-1]==0)
{
      road[c][l]=0;
      road[c+2][l-1]=i;
      engine[i].pos=c+2;
      engine[i].lane=l-1;
      engine[i].speed=2;
      engine[i].ok=1;
      engine[i].shiftornot=1;
```

```
}

};


void car::offrampexit(car engine[],int i,int t)
{
int c=engine[i].pos;
int l=engine[i].lane;

            if(engine[i].speed<5)
                engine[i].speed++;
                {
                 road[c][l]=0;
                    engine[i].pos+=engine[i].speed;
                    c=engine[i].pos;
                    road[c][l]=i;
                    engine[i].lane=l;
                    engine[i].speed=engine[i].speed;
                    engine[i].ok=1;
              }
                     if(c>=OFFRAMP)
                       {
                        road[c][l]=0;
                          engine[i].exit=1;
                       }
      };

void car::adaptspeed(struct car engine[],int i,int t)
{
      int c=engine[i].pos;
      int l=engine[i].lane;
   int r=myrandom(100);
      int decprob=10;
   int z=c+engine[i].speed-1;
      if(r<decprob && engine[i].speed>1 && road[z][l]==0)
                engine[i].speed--;
      else if( engine[i].frontdistance>=5)
      {
if(engine[i].speed<5)
engine[i].speed++;
}
else
{
                if(engine[i].speed>engine[i].frontdistance)
                          engine[i].speed=engine[i].frontdistance;
                else if (engine[i].speed +1<= engine[i].frontdistance &&  engine[i].speed<5)
                          engine[i].speed++;
}
```

```
road[c][l]=0;
engine[i].pos+=engine[i].speed;
c=engine[i].pos;
road[c][l]=i;
engine[i].ok=1;
};


void car::makecellforaux(struct car engine[],int i)
{
int c=engine[i].pos,makecell=0;
int l=engine[i].lane,d=1;
for(int s=c+1;s<=c+5;s++){
      if(road[s][l-1]!=0){
            int n=road[s][l-1];
      if( engine[n].speed>engine[i].frontspeed+1 ||  engine[n].speed > engine[i].speed+1)
{
      int gplus=engine[n].pos-engine[i].pos-d;         //distance between the cars  RULE 1
      int D=d+engine[i].speed;                         //safe gap
if(gplus > min(engine[i].speed,D-d))
//Dplus and Dminus can be used proportional to preceding cars speed,too.
      {
        for(int m=c-1;m>c-5;m--)       //looks for the trailing car on the main road.
            if(road[m][l-1]!=0)
            {
                  int u=road[m][l-1];            //trailing car at right
                  int gminus=engine[i].pos-engine[u].pos-d;
                  D=d+engine[i].speed;
                  if(gminus > min(engine[u].speed,D-d))
makecell:         {
                        road[c][l]=0;
                        if(road[s+1][l-1]==0 && engine[i].speed<5)
                              engine[i].speed++;
                        else
                              engine[i].speed=engine[i].speed;
                              c+=engine[i].speed;
                              road[c][l-1]=i;//new position after lane changing.

                          engine[i].pos=c;
                              engine[i].lane=l-1;
                              engine[i].shiftornot=1;
                              engine[i].ok=1;
                              break;
                              }
                        }
                  }
            }
      }else if (s==c+4)
            goto makecell;
```

```
if(engine[i].shiftornot!=0)
            break;
}
};


void car::onramp(struct car engine[],int i,int t)
{
      int c=engine[i].pos;
      int l=engine[i].lane,s=0;

if(engine[i].speed<5 && c+1+engine[i].speed>=ONRAMPEND || engine[i].speed==5 &&
c+engine[i].speed>=ONRAMPEND)
{
      for(s=ONRAMPEND;s<ONRAMPEND+5;s++)
    //ıt looks where the preceding car is on the target lane.
{
       if(road[s][1]!=0){
          int f=road[s][1];
           if(engine[f].pos>c+engine[i].speed)//safe distance for lane changing.
           {
                 road[c][ONRAMPLANE]=0;
                engine[i].pos+=engine[i].speed;
                c=engine[i].pos;
                road[c][1]=i;//new position after lane changing.
                engine[i].lane=1;
                engine[i].speed=engine[f].speed;
                engine[i].ok=1;
                break;
           }
          else
          {
              engine[i].speed=(engine[f].pos-engine[i].pos-1);
                        if(engine[i].speed+c>=ONRAMPEND)
                        {
                                  road[c][ONRAMPLANE]=0;
                                  engine[i].pos+=engine[i].speed;
                                  c=engine[i].pos;
                                  road[c][1]=i;//new position after lane changing.
                                  engine[i].lane=1;
                                  engine[i].speed=engine[f].speed;
                                  engine[i].ok=1;
                                  break;
                        }
                        else
                        {
                                  road[c][ONRAMPLANE]=0;
                                  engine[i].pos+=engine[i].speed;
```

```
                        c=engine[i].pos;
                        road[c][ONRAMPLANE]=i;//new position after lane changing.
                        engine[i].lane=0;

                    if(engine[i].speed<5)
                            engine[i].speed++;
                        else
                            engine[i].speed=engine[i].speed;
                            engine[i].ok=1;
                            break;
                    }
                     }
            }else if(s==ONRAMPEND+4)
                    {
                 if(engine[i].speed<5)
                            engine[i].speed++;
                            road[c][ONRAMPLANE]=0;
                            engine[i].pos+=engine[i].speed;
                            c=engine[i].pos;
                            road[c][1]=i;//new position after lane changing.
                            engine[i].lane=1;
                            engine[i].ok=1;
                            break;
            }
            }
            }else
                {
                    if(engine[i].speed<5)
                            engine[i].speed++;
                            road[c][l]=0;
                            c+=engine[i].speed;
                            engine[i].pos=c;
                            l=engine[i].lane;
                            road[c][l]=i;
                            engine[i].ok=1;
            },
};

void car::nearofframp(struct car engine[],int i,int t)
{
  int c=engine[i].pos;
  int l=engine[i].lane;
  int d=1,e=0,s=0,mis=0;
       {
       for(s=c+1;s<=c+5;s++) //look at the right lane,may be you should go faster.
         {
            if(road[s][l-1]!=0)                         //preceding car at right
            {
                    e=road[s][l-1];
```

```
                int gplus=engine[e].pos-engine[i].pos-d;
                        if(c+engine[i].speed<engine[e].pos-1)
                        {
                            if(engine[i].speed<5)
                                engine[i].speed++;
                        }
                        else if(c+engine[i].speed>=engine[e].pos)
                        {
                              int p=(engine[e].pos-c-engine[i].speed);
                              if(p>=1)
                                        engine[i].speed=p;
                              else
                                 {  engine[i].speed=engine[i].speed;
                                   goto mis;
                                 }
                         }
                              road[c][l]=0;
                              c+=engine[i].speed;
                              road[c][l-1]=i;//new position after lane changing.
                              engine[i].pos=c;
                              engine[i].lane=l-1;
                              engine[i].shiftornot=1;
                              engine[i].ok=1;
                              engine[i].speed=engine[e].speed;
                              goto mis;
                              }
                    }
                if(s==c+5)
                    {
                          if(engine[i].speed<5)
                                    engine[i].speed++;
                                    road[c][l]=0;
                                   c+=engine[i].speed;
                                   road[c][l-1]=i;//new position after lane changing.
                                    engine[i].pos=c;
                                    engine[i].lane=l-1;
                                    engine[i].shiftornot=1;
                                    engine[i].ok=1;
                                    engine[i].speed=engine[e].speed;
                                    goto mis;
                                    }
                              }
mis: int h=0;}

void car::aux(struct car engine[ ],int i,int t)
{
      int c=engine[i].pos;
    int l=engine[i].lane,s=0;
```

```
    if(engine[i].speed<5 && c+1+engine[i].speed>=AUXLANEMERGE || engine[i].speed==5 && c+engine[i].speed>=AUXLANEMERGE)
{

for(s=AUXLANEMERGE;s<AUXLANEMERGE+5;s++)
//ıt looks where the preceding car is on the target lane.
{
    if(road[s][4]!=0){
            int f=road[s][4];
            if(engine[f].pos>c+engine[i].speed)    //safe distance for lane changing.
            {
                    road[c][AUXLANE]=0;
                    engine[i].pos+=engine[i].speed;
                    c=engine[i].pos;
                    road[c][4]=i;//new position after lane changing.
                    engine[i].lane=4;
                    engine[i].speed=engine[f].speed;
                    engine[i].ok=1;
                    break;
            }
            else
            {
                engine[i].speed=(engine[f].pos-engine[i].pos-1);
                    if(engine[i].speed+c>=AUXLANEMERGE)
                    {
                            road[c][AUXLANE]=0;
                            engine[i].pos+=engine[i].speed;
                            c=engine[i].pos;
                            road[c][4]=i;//new position after lane changing.
                            engine[i].lane=4;
                            engine[i].speed=engine[f].speed;
                            engine[i].ok=1;
                            break;
                    }
                    else
{
        road[c][AUXLANE]=0;
        engine[i].pos+=engine[i].speed;
        c=engine[i].pos;
        road[c][5]=i;//new position after lane changing.
        engine[i].lane=5;
    if(engine[i].speed<5)
            engine[i].speed++;
  else
            engine[i].speed=engine[i].speed;
            engine[i].ok=1;
           break;
}
}
```

```
}
else if(s==AUXLANEMERGE+4)
            {
      if(engine[i].speed<5)
            engine[i].speed++;
            road[c][AUXLANE]=0;
            engine[i].pos+=engine[i].speed;
            c=engine[i].pos;
            road[c][4]=i;//new position after lane changing.
            engine[i].lane=4;
            engine[i].ok=1;
            break;
}
            }
                  }else
                  {
                        if(engine[i].speed<5)
                              engine[i].speed++;
                              road[c][l]=0;
                              c+=engine[i].speed;
                              engine[i].pos=c;
                              l=engine[i].lane;
                              road[c][l]=i;
                              engine[i].ok=1;
                  }
};

void car::onrampadaptspeed(struct car engine[ ],int i,int t)
{
      int c=engine[i].pos;
      int l=engine[i].lane;
      int r=myrandom(100);
      int decprob=10;
   int z=c+engine[i].speed-1;
      if(r<decprob && engine[i].speed>1 && road[z][l]==0)
            engine[i].speed--;
      else if( engine[i].frontdistance>=5 && c+1+engine[i].speed < ONRAMPEND)
      {
         if(engine[i].speed<5)
              engine[i].speed++;
   }
      else
      {
           if(engine[i].speed>engine[i].frontdistance )
               engine[i].speed=engine[i].frontdistance;
           else if(engine[i].speed+1<= engine[i].frontdistance &&  engine[i].speed<5 &&
c+1+engine[i].speed < ONRAMPEND)
           engine[i].speed++;
      }
```

```
                    if(engine[i].speed+c >=ONRAMPEND)
                engine[i].speed=ONRAMPEND-(engine[i].speed+c);
            if(engine[i].speed<1)
                engine[i].speed=0;
             road[c][l]=0;
               engine[i].pos+=engine[i].speed;
               c=engine[i].pos;
               road[c][l]=i;
               engine[i].ok=1;
  };


void car::lanechangecareless(struct car engine[ ],int i,int t)
{
      int gminus=0,gplus=0;
      {
      int f=0,e=0,z=0,d=1,x=0,shiftleft=0,shiftright=0,speedgap=0;
      int c=engine[i].pos,decision=0;
      int l=engine[i].lane,lookright=0,s=0,rightvel=0,leftvel=0;

      if(engine[i].frontdistance < engine[i].speed || engine[i].frontspeed<=engine[i].speed )
      {
              if(l==1 || l==2 || l==3 )
            {
                 for(s=c+1;s<=c+5;s++)
                       {
                              if(road[s][l+1]!=0)
                              {
                                       int f=road[s][l+1];
                                       int gplus=s-engine[i].pos-d;
                                       if( gplus > engine[i].frontdistance)
                                       {
                                     if(c+engine[i].speed<s-1)
                                              {
                                         if(engine[i].speed<5)
                                                      engine[i].speed++;
                                       }
                                              else if(c+engine[i].speed>s-1)
                                                {
                                         engine[i].speed=(s-engine[i].pos-1);
                                                 if(engine[i].speed<1)
                                                 {
                                         engine[i].speed+=(engine[i].speed+c-s-1);
                                                  goto out;
                                          }
                                          }
leftvel=engine[i].speed;
shiftleft=engine[i].pos+engine[i].speed;
    goto lookright;
```

```
    }
                                        else goto lookright;}
                                        }
    if(s==c+6)
    {   if(engine[i].speed<5)
                                                engine[i].speed++;
                                                shiftleft=engine[i].pos+engine[i].speed;
                                                leftvel=engine[i].speed;
}

lookright:              if(l==2 || l==3)
                        {int s=0;
                        for(s=c+1;s<=c+5;s++)
                        //look at the right lane,may be you should go faster.
                        {
    if(road[s][l-1]!=0)     //preceding car at right
                                {
        int e=road[s][l-1];
                                        int gplus=s-engine[i].pos-d;
                                        if( gplus > engine[i].frontdistance)
                                        {
    if(c+engine[i].speed<s-1)
                                                {
    if(engine[i].speed<5)
                                                        engine[i].speed++;
    }
                                    else if(c+engine[i].speed>s-1)
                                    engine[i].speed=(s-engine[i].pos-1);
                                    if(engine[i].speed<1)
                                    {
    engine[i].speed+=(engine[i].speed+c-s-1);
                                        goto out;
    }
                                rightvel=engine[i].speed;
                                shiftright=engine[i].pos+engine[i].speed;
                                goto decision;
                                }
                                else goto decision;}
                                }if(s==c+6)
                                {
                                   if(engine[i].speed<5)
                                        engine[i].speed++;
                                        shiftright=engine[i].pos+engine[i].speed;
                                        rightvel=engine[i].speed;
                                        goto decision;
                                }
                               }
                                else goto decision;
decision:                       if(shiftright>shiftleft)
```

```
                        {
                         {
road[c][l]=0;
                                        c+=rightvel;
                                        engine[i].speed=rightvel;
                                         road[c][l-1]=i;//new position after lane changing.
                                        engine[i].pos=c;
                                        engine[i].lane=l-1;
                                        engine[i].shiftornot=1;
                                        engine[i].ok=1;
                                        rightgone++;
                                        goto out;
     }
                                  //else goto out;
                        }
                        else if(shiftright<shiftleft)
                        {
                       {
                          road[c][l]=0;
                               c+=leftvel;
                              engine[i].speed=leftvel;
                              road[c][l+1]=i;//new position after lane changing.
                               engine[i].pos=c;
                               engine[i].lane=l+1;
                              engine[i].shiftornot=1;
                               engine[i].ok=1;
                               leftgone=i;
                               goto out;
                        }
                        }
else if(shiftleft==shiftright && shiftleft!=0 && shiftright!=0)
{
                          int y=myrandom(100);
                          if(y<50)
                         {
                             shiftleft++;
                                  else shiftright++;
                                  goto decision;
                      }
                      else if(shiftleft==0 && shiftright==0)
                                  goto out;
                }
  if(l==4)
  {
int s=0;
  for(s=c+1;s<=c+5;s++)
  {
  if(road[s][l-1]!=0)
    {
```

```
        int e=road[s][l-1];
          int gplus=s-engine[i].pos-d;
          if( gplus > engine[i].frontdistance)
              {
   if(c+engine[i].speed<s-1)
                      {
   if(engine[i].speed<5)
                              engine[i].speed++;}
                              else if(c+engine[i].speed>s-1)
                              {
                                      engine[i].speed=(s-engine[i].pos-1);
                                      if(engine[i].speed<1)
                                      {
        engine[i].speed+=(engine[i].speed+c-s-1);
                                              goto out;
        }
}
                              shiftright=c+engine[i].speed;
                              rightvel=engine[i].speed;
                              goto decision;
   }
                      else goto decision;
     }
     }if(s==c+6)
{
                if(engine[i].speed<5)
               engine[i].speed++;
               shiftright=c+engine[i].speed;
               rightvel=engine[i].speed;
               goto decision;
     }
out:int w=0 ;
}
}else goto out;
}
};

void car::lanechangeadvantage(struct car engine[],int i,int t){
               int gminusleft=0,gplusleft=0,gplusright=0,gminusright=0;
               int f=0,e=0,z=0,d=1,x=0,shiftleft=0,shiftright=0,speedgap=0;
               int c=engine[i].pos;
               int decision=0;
               int l=engine[i].lane,out=0,s=0,goleft=0,goright=0,other=0,lookright=0;

               if(l!=4)
               {
               for(s=c+1;s<=c+20;s++)
               {
               if(s>=499)
```

```
            {gplusleft=s-engine[i].pos-d;
            if(gplusleft>=5)
            {
            engine[i].advantage=myrandom(0.20)+0.80;
    // probability between 0.80 and 1.00   very advantageous
            if((engine[i].will*engine[i].advantage)>=0.200 && engine[i].will> 0.20 )
                    goto lookright;
            else goto other;
            }
            }
            if(road[s][l+1]!=0)
                    {
                            f=road[s][l+1];//at left
                            gplusleft=s-engine[i].pos-d;
                            break;
                    }
            }
            }

lookright:          if(l!=1)
                      {       for(s=c+1;s<=c+20;s++)
                              {
                              if(s>=CELL)
                              {
        gplusright=s-engine[i].pos-d;
                                      if(gplusleft>=5 || gplusright>=5)
                                      {
        engine[i].advantage=myrandom(0.20)+0.80;
// probability between 0.80 and 1.00   very advantageous
     ((engine[i].will*engine[i].advantage)>=0.200 && engine[i].will> 0.20 )
            goto decision;
            else
            goto other;
            }
                    }
                            if(road[s][l-1]!=0)
                            {
                                    e=road[s][l-1]; // at right
                                    gplusright=s-engine[i].pos-d;
                                    break;
                            }
                     }
            }

if(engine[i].speed==5 && engine[f].speed==5 && f!=0 && engine[i].frontdistance <
engine[f].pos-c-1 && l!=4)//for left
 {
        engine[i].advantage=myrandom(0.20)+0.60;
        //with prob 0.60 and 0.80 advantageous
```

```
            if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.4)
                goto decision;
 }

if(engine[i].speed==5 && engine[e].speed==5 && e!=0 && engine[i].frontdistance < engine[e].pos-c-1 && l!=1) //for right
{
        engine[i].advantage=myrandom(0.20)+0.60;
        //with prob 0.60 and 0.80 advantageous
           if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.4     )
              goto decision;
}
other:        if(gplusright>engine[i].frontdistance || gplusleft>engine[i].frontdistance)
              {
              if(gplusleft>=5 || gplusright>=5)
              {
        engine[i].advantage=myrandom(0.20)+0.80;
        // probability between 0.80 and 1.00   very advantageous
   if((engine[i].will*engine[i].advantage)>=0.200 && engine[i].will> 0.20 )
                        goto decision;
              }
                    if(engine[i].frontdistance < engine[i].speed)
                    {
                        if( engine[i].frontspeed<=engine[i].speed )
                        {
        engine[i].advantage=myrandom(0.20)+0.80;
        // probability between 0.80 and 1.00   very advantageous

     if((engine[i].will*engine[i].advantage)>=0.200 && engine[i].will> 0.20 )
     {
decision:             if(gplusright>gplusleft)
    {
goright:               if(engine[i].speed<=gplusright)
                        {
                        road[c][l]=0;
                        engine[i].pos+=engine[i].speed;
                        c=engine[i].pos;
                        road[c][l-1]=i;//new position after lane changing.
                        engine[i].lane=l-1;
                        engine[i].shiftornot=1;
                            engine[i].ok=1;
                        r++;
                        goto out;
                        }
else if(gplusright!=0)
                      {
   engine[i].speed=gplusright;
                            goto goright;
                        }
```

```
                    }
    else if(gplusright<gplusleft)
                    {
goleft:                     if(engine[i].speed<=gplusleft)
                            {
                                    road[c][l]=0;
                                    engine[i].pos+=engine[i].speed;
                                    c=engine[i].pos;
                                    road[c][l+1]=i;//new position after lane changing.
                                    engine[i].lane=l+1;
                                    engine[i].shiftornot=1;
                                    engine[i].ok=1;
                                    e++;
                                    goto out;
                                    }
    else if(gplusleft!=0)
                            {
                                    engine[i].speed=gplusleft;
                                    goto goleft;
                            }
                    }else
                            if(gplusright==gplusleft && l!=4 && l!=1)
                             {
                                    int y=myrandom(100);
                                    if(y<50)
                                            goto goright;
                                    else
                                            goto goleft;
                             }
                                     else if(gplusright==gplusleft && l==4)
                                            goto goright;
                                    else if(gplusright==gplusleft && l==1)
                                             goto goleft;
                    }
            }else //if the preceding is faster than the vehicle.
                {
        engine[i].advantage=myrandom(0.20)+0.60;
        //with prob 0.60 and 0.80 advantageous
        if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.4)
            goto decision;
            }
     }else  // if headway is bigger than or equals to velocity.
    {
     engine[i].advantage=myrandom(0.20)+0.60;   //with prob 0.60 and 0.80 advantageous
     if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.4)
    goto decision;
     }
}
```

```
else if(gplusleft<engine[i].frontdistance || gplusright < engine[i].frontdistance)
// (gplus left or right is not far from frontdistance)
      {
    engine[i].advantage=myrandom(0.20)+0;          // more disadvadvantageous
            if((engine[i].will*engine[i].advantage)>0.160 && engine[i].will< 0.80)
                        goto decision;
               else if((engine[i].will*engine[i].advantage)>0.160)
               {
                        int y=myrandom(100);//randomly chosen //wrt to driver
                        if(y<50)
                                goto decision;
                        }
                        if(gplusleft==engine[i].frontdistance)   //for left
                        {
                                if(engine[f].speed>engine[i].frontspeed)
                                {
                                        if(engine[f].speed>engine[i].speed)
                                        {
                                        engine[i].advantage=myrandom(0.20)+0.60;
    // advantageous 0.60 between 0.80
                        if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.40)
                                        goto goleft;
else      if(engine[i].will>0.40      &&      (engine[i].will*engine[i].advantage)>0.160      &&
(engine[i].will*engine[i].advantage)<0.360)
{
int y=myrandom(100);
if(y>50)
goto goleft;
                        }
               }
               else if(engine[f].speed<engine[i].speed)
               {
    engine[i].advantage=myrandom(0.20)+0.20;     // disadvantageous
                      if((engine[i].will*engine[i].advantage)>0.160)
                                goto goleft;
                        else if((engine[i].will*engine[i].advantage)>0.120)
                        {
                                int y=myrandom(100);
                                if(y>50)
                                        goto goleft;
                        }
               }
               else if(engine[f].speed==engine[i].speed)
               if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.40)
                        goto goleft;
          else    if(engine[i].will>0.40    &&    (engine[i].will*engine[i].advantage)>0.160    &&
(engine[i].will*engine[i].advantage)<0.360)
               {
    int y=myrandom(100);
```

```
                    if(y>50)
                    goto goleft;
 }
     }else if(engine[f].speed<engine[i].frontspeed)
    {
        engine[i].advantage=myrandom(0.20)+0.00;
        // most disadvantageous 0.00 and 0.20
          if(engine[i].will>0.80)
            {
                    int y=myrandom(100);//randomly chosen //wrt to driver
                    if(y>50)
                    goto decision;
            }
     }
else if(engine[f].speed==engine[i].frontspeed)
     {
   engine[i].advantage=myrandom(0.2)+0.4;       // no need
      if(engine[i].will>0.60 && (engine[i].will*engine[i].advantage)>0.240)
        goto goleft;
      else    if(engine[i].will>0.40    &&    (engine[i].will*engine[i].advantage)>0.160    &&
(engine[i].will*engine[i].advantage)<0.360)
     {
            int y=myrandom(100);
if(y>50)
                    goto goleft;
                            }
                    }
            }
            else if(gplusright==engine[i].frontdistance)  //for right
            {
                    if(engine[e].speed>engine[i].frontspeed)
                    {
                    if(engine[e].speed>engine[i].speed)
                    {
                        engine[i].advantage=myrandom(0.20)+0.60;
   // advantageous 0.60 between 0.80
                      if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.40)
                            goto goright;
                    else  if(engine[i].will>0.40  &&  (engine[i].will*engine[i].advantage)>0.160
&& (engine[i].will*engine[i].advantage)<0.360)
     {
    int y=myrandom(100);
            if(y>50)
                    goto goright;
     }
     }
            else if(engine[e].speed<engine[i].speed)
            {
                    engine[i].advantage=myrandom(0.20)+0.20;              // disadvantageous
```

```
                        if((engine[i].will*engine[i].advantage)>0.160)
                                goto goright;
                           else if((engine[i].will*engine[i].advantage)>0.120)
                           {
                                int y=myrandom(100);
                                if(y>50)
                                goto goright;
                                                }
                                        }
                        else if(engine[e].speed==engine[i].speed)
                        {
                   if((engine[i].advantage*engine[i].will)>0.240 && engine[i].will>=0.40)  //Advantageous
                        goto goright;
                     else  if(engine[i].will>0.40  &&  (engine[i].will*engine[i].advantage)>0.160  &&
                (engine[i].will*engine[i].advantage)<0.360)
                     {
                   int y=myrandom(100);
                        if(y>50)
                                goto goright;
                    }
                     }
                     }else if(engine[e].speed<engine[i].frontspeed)
{
engine[i].advantage=myrandom(0.20)+0.00;
// most disadvantageous 0.00 and    0.20
                if(engine[i].will>0.80)
                     {
                        int y=myrandom(100);//randomly chosen //wrt to driver
                        if(y>50)
                        goto decision;
                     }
                else if(engine[e].speed==engine[i].frontspeed)
                        {
                                engine[i].advantage=myrandom(0.2)+0.4;      // no need
                               if(engine[i].will>0.60                                  &&
                        (engine[i].will*engine[i].advantage)>0.240)                    goto
                        goright;
                                else                    if(engine[i].will>0.40          &&
                           (engine[i].will*engine[i].advantage)>0.160                   &&
                           (engine[i].will*engine[i].advantage)<0.360)
                                {
                                        int y=myrandom(100);
                                        if(y>50)
                                    goto goright;
                                }
                                }
                                }
                                }
                        out: int w=0;
```

```
}

 void car::vacancy(struct car engine[],int i,int t)
{
        int c=engine[i].pos,rightpos=0,leftpos=0,out=0;
   int l=engine[i].lane,j=0,k=0,f=0,right=0,left=0;
        {
                for(int b=c+1;b<=c+6;b++)
           {
                        if(road[b][l+1]!=0 && l!=4 && l!=0)
                        {
                           j=road[b][l+1];
                           if(engine[i].speed==5 && engine[j].speed==5
                           && engine[i].frontdistance < engine[j].pos-c-
                           1)
                                        leftpos=engine[j].pos;
                     }
                        if(road[b][l-1]!=0 && l!=1)
                        {
                           k=road[b][l-1];
                           if(engine[i].speed==5 && engine[k].speed==5
                           && engine[i].frontdistance < engine[k].pos-c-
                           1)
                                rightpos=engine[k].pos;}


        if(leftpos>rightpos)
        {
left:   engine[i].pos+=engine[i].speed;
                road[c][l]=0;
                c=engine[i].pos;
                road[c][l+1]=i;
                engine[i].lane=l+1;
                engine[i].ok=1;
                engine[i].shiftornot=1;
                engine[i].speed=5;
                break;
        }
        else if(leftpos<rightpos)
        {
right:  engine[i].pos+=engine[i].speed;
                road[c][l]=0;
                c=engine[i].pos;
                road[c][l-1]=i;
                engine[i].lane=l-1;
                engine[i].ok=1;
                engine[i].shiftornot=1;
                engine[i].speed=5;
                break;
```

```
        }
        else if(leftpos==rightpos && leftpos!=0 && rightpos!=0)
        {
            f=myrandom(100);
                if(f<50 && l!=4)
                        goto left;
                else if(l!=0)
                        goto right;
                else
                        goto out;
                }else
                    goto out;
}
        }
out: int p=0; }

int main()
        {
        struct car car[NUMBER];
        int i=1,t=0,c=0,l=0,j=0;
        int speedgap=0;             //speed gap between the car and its
preceding's.
        float density=0.0;          // initially no cars on the main road.
        int d=1,may=0,D=0,gminus=0,gplus=0;
                                        int  safedistance=0;            //safe
distance that should be leaved when adapting speed.
        int shiftright=0;           //car number that shifts right
        int shiftleft=0;  //car number that shifts left.
        int totalcar=0;   // how many cars on the road.
        int initialcarnum=0;     //used for assigning car no,for following
car number.
        int first=0;              //at beginning of the road,first car finder
        int auxlanecarnum=0;

   int waitingcar=0;     //initially,number of cars that waits for entering
on the onramp.          int add=0,speed=0;
        int again=0,up=0,over=0;

        FILE *Data;           //the file saves initial values.
        FILE  *map;                   //the file shows photo of the main
road,auxiliary road and onramp
        FILE *carenterance;
        FILE *auxlane;       //exiting cars from  onramp and their times.
        FILE *onramp;        //exiting cars from mainroad to offramp and
their times
        FILE *offramp;
        FILE *mainroad;
        FILE *detector;
        FILE *xtgraph;
```

```
        FILE *densitytime;
        FILE *control;

if ((densitytime = fopen("densitytime.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }
if ((Data = fopen("Data.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }

if ((control = fopen("control.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }
if ((map= fopen("map.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }

 if ((auxlane= fopen("auxlane.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }
if ((onramp = fopen("onramp.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }
if ((offramp = fopen("offramp.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }
if ((mainroad = fopen("mainroad.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }
if ((detector = fopen("detector.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
```

```
        }
if ((carenterance = fopen("carenterance.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }

if ((xtgraph= fopen("xtgraph.txt", "w")) == NULL)
        {
                printf("File did not open...\n");
                exit(1);
        }

for(int u=1;u<=NUMBER;u++)
{//at random,loop puts cars onto the main road until the capacity of the
road is exceeded.
int   c=myrandom(CELL-1);
int   l=myrandom((LANE-1)+1);
type2carcome=((1500*INITIALDENSITY*type2dense)/(30*100));
type1carcome=((1500*INITIALDENSITY*type1dense)/(30*100));
if(road[c][l]==0)
{
        if(c<OFFRAMP)
        {
           if((int)(carnum*100/1500) < INITIALDENSITY)
{
if((c>=AUXLANEMERGE-1 && c<AUXLANEMERGE+5 && l==3)
|| l==0 || (c>AUXLANEMERGE && l==5) || (l==5 &&
c<=AUXLANEMERGE &&
auxlanecarnum*100/350>=AUXLANEDENSITY))
//if the lane is 3, after auxlane,do not put cars in order to simulate the
shifting of cars on 3 to lane2 initially.
        {
                road[c][l]=0;
                u--;
                }
else
{
if( c<OFFRAMP-10 && type2carnum < type2carcome && l!=5 &&
l!=0)
{   //if cars position is after the tenth cell,define as type2.
                type2carnum++;
                car[u].type=2;}
        else if(l==AUXLANE && type1carnum < type1carcome )
{
                type1carnum++;
                car[u].type=1;
                auxlanecarnum++;
}
```

```
 else if (type1carnum < type1carcome )
{
                    type1carnum++;
                    car[u].type=1;}
                    car[u].pos=c;
                    car[u].lane =l;
                    car[u].entry=car[u].pos;
                    car[u].shiftornot=0;
                    car[u].speed=(myrandom(MAXSPEED-
MINSPEED)+MINSPEED);
            //at random speed
                    car[u].exit=0;
                    road[c][l]=u;
                    carnum++;
                    initialcarnum++;
                    car[u].ok=0;
                    fprintf(Data,"%d.car          %d.pos
        %d.lane              %d speed              %d
type\n",u,car[u].pos,car[u].lane,car[u].speed,car[u].type);}
                    }
            else break;
                    }
              else
              {
                        if((c>=AUXLANEMERGE-1 &&
c<AUXLANEMERGE+5 && l==3) || l==0 || (c>AUXLANEMERGE
&& l==5) || (l==5 && c<=AUXLANEMERGE &&
auxlanecarnum*100/350>=AUXLANEDENSITY)){ //if the
lane is 3, after auxlane,do not put cars in order to simulate the shifting
of cars on 3 to lane2 initially.
road[c][l]=0;
        u--;
}//if this position is not empty
else
{
        int y=myrandom(100);
   if(y<50)
        {
            car[u].type=1;
                    car[u].pos=c;
                    car[u].lane =l;
                    car[u].shiftornot=0;
            car[u].entry=car[u].pos;
                    car[u].speed=(myrandom(MAXSPEED-
MINSPEED)+MINSPEED);
             //at random speed
                    car[u].exit=0;
                    road[c][l]=u;
                    totalcar++;
```

```
                initialcarnum++;
                car[u].ok=0;
                fprintf(Data,"%d.car          %d.pos
        %d.lane              %d speed              %d
type\n",u,car[u].pos,car[u].lane,car[u].speed,car[u].type);}
                else
                        u--;
        }
        }
        }else u--;
}

for(u=(initialcarnum+1);u<=NUMBER;u++)
 {    //loop puts the cars onto the onramp
if(waitingcar*100/30<ONRAMPDENSITY)
//until the capacity of the auxiliary road is exceeded,put cars on the
onramp.
{
        int                          w=(myrandom(ONRAMPEND-
ONRAMPBEGIN)+ONRAMPBEGIN);
//to the random position on onramp road.
        if(road[w][ONRAMPLANE]==0)
        {
                car[u].pos=w;
                car[u].type=1;
                car[u].lane=ONRAMPLANE;
                car[u].entry=car[u].pos;
                car[u].speed=(myrandom(MAXSPEED-
MINSPEED)+MINSPEED);
            //with random speed
                waitingcar++;
                initialcarnum++;
                car[u].exit=0;
                car[u].shiftornot=0;
                road[w][ONRAMPLANE]=u;
                car[u].ok=0;
        fprintf(Data,"%d.car          %d.pos              %d.lane
                %d                                          speed
        %d.type\n",u,car[u].pos,car[u].lane,car[u].speed,car[u].type);
//to save number of cars on the onramp.
        }
        else
                u--;
}
else
break;
}
for(c=CELL-1;c>=0;c--)
```

```
fprintf(Data,"%d %d %d %d
	%d %d %d
\n",c,road[c][5],road[c][4],road[c][3],road[c][2],road[c][1],road[c][0]);

for(t=1;t<=ITER;t++)
{   //MAIN LOOP
 int gonetype1=0;
 int gonetype2=0;
 for(int w=1;w<=initialcarnum;w++)
	car[w].ok=0;
 for(l=LANE-2;l>=0;l--)
{
	for(int c=CELL-1;c>=0;c--)
	{
		if(road[c][l]!=0)
		{
			i=road[c][l];
			c=car[i].pos;
			l=car[i].lane;
			int f=c;
			int p=myrandom(100);
			car[i].shiftornot=0;
			if(car[i].exit==0)
			{
				if(car[i].ok==0)
				{
					if(l!=0)
						car[i].inspectfront(car,i,t);
					else if(l==0)

	car[i].onrampinspectfront(car,i,t);
if(car[i].type==2 && c<OFFRAMP && ( (c>=OFFRAMP-7 && l==2)
|| (c>=OFFRAMP-9 && l==3) || (c>=OFFRAMP-11 && l==4)))
	{
	car[i].headoff(car,i,t);
		if(car[i].shiftornot==0 && road[c+1][l]==0)
		 {
		   car[i].speed=1;
		   road[c][l]=0;
		   car[i].pos+=car[i].speed;
		   car[i].ok=1;
			car[i].lane=l;
			c=car[i].pos;
			road[c][l]=i;
		   }
		   else if(car[i].ok==0)
			car[i].adaptspeed(car,i,t);
	}
```

```
else if(car[i].type==2 && c<OFFRAMP && (l==2 && c>=OFFRAMP-15 || l==3 && c>OFFRAMP-25 || l==4 && c>OFFRAMP-35))
{
		car[i].nearofframp(car,i,t);
		if(car[i].shiftornot==1)
		fprintf(offramp,"%d.car shifts to %d at %d \n",i,car[i].lane,t);
		else if(car[i].shiftornot==0)
		{
			car[i].lanechangecareless(car,i,t);
			if(car[i].shiftornot==0)
			car[i].adaptspeed(car,i,t);
			if(car[i].speed==0 && car[i].frontspeed==0)
			int y=0;
		}
}

	else if(car[i].type==2 && car[i].lane==1 && car[i].front==0)
	{
		car[i].offrampexit(car,i,t);
		if(car[i].exit==1)
		fprintf(offramp,"%d.car->offramp  %d.second\n",i,t);
	}
	   //else if(car[i].type==2 && car[i].pos> 300 && l==1)
	   //int y=0;
	else if(l==ONRAMPLANE)    //for the car is onramp but is not in the onrapm merging region
{
		if(car[i].front==0)
		{
		    car[i].onrampinspectfront(car,i,t);
			car[i].onramp(car,i,t);
			if(car[i].lane==1)
			{
			waitingcar--;
			fprintf(onramp,"%d enters from ONRAMP at %d second \n",i,t);
			}
		}
		else if(car[i].ok==0)
		 {
		  car[i].onrampinspectfront(car,i,t);
			car[i].onrampadaptspeed(car,i,t);
		}
	}
else	if(l==4	&&	c>AUXLANEMERGE-15	&&	c<AUXLANEMERGE+10)
```

```
 //loop is used for shifting the cars on the lane3 to right if the position is before the auxlane.Cars should shift right in order to enter the cars on auxlane to the main road.
{
                car[i].makecellforaux(car,i);
                if(car[i].lane==3)
                fprintf(auxlane,"%d.car shifts from 4 to 3.lane at %d.second\n",i,t);
                else
                        car[i].adaptspeed(car,i,t);
}
else if(car[i].front!=0 && l!=0 && l!=5 && car[i].ok==0 )
        {
                if(car[i].type==2 && car[i].pos<OFFRAMP-20 || car[i].type==1)
                {
                        car[i].lanechangecareless(car,i,t);
                        if(leftgone!=0)
                        {
                           fprintf(auxlane,"%d shifts left at %d\n",leftgone,t);
                                leftgone=0;
                         }
                                if(car[i].shiftornot==0 && car[i].speed==5)
                                        car[i].vacancy(car,i,t);
                                if(car[i].shiftornot==0 && car[i].ok==0)
                                        car[i].adaptspeed(car,i,t);
                }
                                else if(car[i].shiftornot==0)
                                        car[i].adaptspeed(car,i,t);
}
else if(car[i].ok==0 && car[i].shiftornot==0 && car[i].front!=0 )
                                        car[i].adaptspeed(car,i,t);

else if(car[i].front==0 &&  car[i].ok==0 && car[i].shiftornot==0 )
{
        if(car[i].speed<5)
                car[i].speed++;
        else
                car[i].speed=car[i].speed;
                road[c][l]=0;
                car[i].pos+=car[i].speed;
                int e=car[i].pos;
                road[e][l]=i;
                car[i].ok=1;
}
        if(car[i].pos>=CELL && (car[i].lane!=AUXLANE || car[i].lane!=ONRAMPLANE))
```

```
        {
int k=car[i].pos-car[i].speed;
road[k][l]=0;
int y=car[i].pos;
road[y][l]=0;
car[i].exit=1;
}
if(car[i].ok==0)
        int y=0;
if(f<OFFRAMP && car[i].pos>=OFFRAMP && car[i].ok==1)
{
                if(car[i].type==1)
                {
                    gonetype1++;
                        fprintf(Data,"%d    %d \n",i,t);
                }
                else if(car[i].type==2)
                {
                        gonetype2++;
                        fprintf(Data,"%d   %d\n",i,t);
                        f=0;
                }
                }
        }
    }
  }
 }
}
         {
            l=5;
                c=0;

                 for(int c=AUXLANEMERGE;c>=0;c--)
                 if(road[c][l]!=0)
                 {
                        i=road[c][l];
                        int auxentry=car[i].pos;
                        c=car[i].pos;
                        l=car[i].lane;
                        int f=c;
                        int k=car[i].pos;
                        int p=car[i].lane;
                        car[i].shiftornot=0;
                        if(car[i].exit==0)
                        {
                                if(car[i].ok==0)
                                {
                                        car[i].inspectfront(car,i,t);
                                        if(car[i].front==0)
```

```
                        {
                                car[i].aux(car,i,t);
                                road[k][p]=0;
                                if(car[i].lane==4)
                                fprintf(auxlane,"%d.car shifts from
auxlane to mainroad at %d.second\n",i,t);
                         }
                         else if(car[i].ok==0)
                                 car[i].adaptspeed(car,i,t);
                     if(car[i].ok==0)
                                 int y=0;
                    if(f<OFFRAMP && car[i].pos>=OFFRAMP)
                        {
                                gonetype1++;
                                f=0;
                                fprintf(Data,"%d   %d\n",i,t);
                         }
                        }
                      }
                    }
                  }

c=0;
          for(c=0;c<CELL;c++)
{
   if(road[c][4]!=0)
         {
             int w=road[c][4];
                  if(car[w].type==2 && car[w].entry<40){
                 int s=car[w].pos;
                  int g=car[w].lane;
          }
 }
 }

for(i=1;i<initialcarnum;i++)
 {
                 if(i==64 && car[i].exit==0){
                  //if(t==150)
                  //int y=0;
                  int g=0,d=0,r=0,z=0;
                  int c=car[i].pos;
                  int l=car[i].lane;
                  car[i].inspectnear(car,i,t);
                  car[i].inspectback(car,i,t);
                  car[i].inspectfront(car,i,t);

                 if(l!=4)
                 {
```

```
                for(int x=1;x<=CELL;x++) //looking for the preceding
car
                {
                        if(c-x==0)
                                g=0;
                        if(road[c-x][l+1]!=0)
                                g=road[c-x][l+1];       //finding which car
there is.
                                break;}
                }else
                {g=0;
                car[g].pos=0;
                car[g].speed=0;}

                if(l!=0)
                {
                        for(int  x=1;x<=CELL;x++)  //looking  for  the
preceding car
                        {
                                if(c-x==0)
                                {d=0;
                                break;}
                                if(road[c-x][l-1]!=0)
                                {d=road[c-x][l-1];      //finding which car
there is.
                                break; }
                        }
                }else
                {
                        d=0;
                        car[d].pos=0;
                        car[d].speed=0;}

                if(l!=4)
                {
                        for(int  x=1;x<CELL-1;x++)  //looking  for  the
preceding car
                        {
                                if(c+x==499)
                                {r=0;
                                break;}
                                if(road[c+x][l+1]!=0)
                                {r=road[c+x][l+1];          //finding  which
car there is.
                                break;}


                        }
                }else
```

```
                {r=0;
                car[r].pos=0;
                car[r].speed=0;}

                if(l!=1  && l!=0)
                {
                        for(int  x=1;x<CELL-1;x++)  //looking  for  the
preceding car
                        {
                        if(c+x==499)
                                z=0;
                        if(road[c+x][l-1]!=0)
                        {
                           z=road[c+x][l-1];       //finding which car there
                        is.
                                break;}
                        }
                        }
                           else
                        {
                        z=0;
                        car[z].pos=0;
                   car[z].speed=0;}

          fprintf(control," %d, %d, %d---%d, %d, %d---%d, %d, %d
%d\n",r,car[r].pos,car[r].speed,car[i].front,car[i].frontpos,car[i].frontspe
ed,z,car[z].pos,car[z].speed,t);
          fprintf(control," %d, %d, %d---%d, %d, %d---%d, %d, %d

%d\n",car[i].nearleft,car[i].nearleftpos,car[i].nearleftspeed,i,c,car[i].spe
ed,car[i].nearright,car[i].nearrightpos,car[i].nearrightspeed,t);
          fprintf(control," %d, %d, %d---%d, %d, %d---%d, %d, %d

%d\n",g,car[g].pos,car[g].speed,car[i].back,car[i].backpos,car[i].backsp
eed,d,car[d].pos,car[d].speed,t);
          fprintf(control,"                                        %d\n",t);
          }
                   }
         c=0;
   for(c=CELL-1;c>=0;c--)
                fprintf(mainroad,"%d %d       %d       %d     %d    %d
           %d
%d\n",c,road[c][5],road[c][4],road[c][3],road[c][2],road[c][1],road[c][0
],t);

  type1carnum=0;
  type2carnum=0;

            for(l=5;l>0;l--)
```

```
    {
        int y=0;
              for(c=OFFRAMP-1;c>=0;c--)
              {
                  if(road[c][l]!=0)
                  {
               i=road[c][l];


                   if(car[i].type==1)
                  {
                type1carnum++;
               //fprintf(map,"%d   %d\n",i,t);
                  }
                  else if(car[i].type==2)
                  {
                 type2carnum++;
                       fprintf(map,"%d   %d\n",i,t);
                  }
                        carnum=type1carnum+type2carnum;
                         density=float(carnum*100/1500);
                  }
         }
 }

 fprintf(carenterance,"density:%f   type1: %f   type2: %f %d.second\n
",(float)density,
(float)type1carnum*100/1500,(float)type2carnum*100/1500,t);
 int  enter=initialcarnum+1;
 int safedistance=0,enterance=0,s=0;

while(density<INITIALDENSITY){
over:
    s=myrandom(6);
    if(s==0)
                goto over;
        for(int c=0;c<CELL;c++)
  {
//look for the first car and when entering adapt speed and position wrt
to the first car.
 if(road[c][s]!=0)
 {
          int first=road[c][s];//farthest downstream car
          if(car[first].pos==0)
               goto over;
        enterance:      if(enter<=NUMBER)
                  {
                       if(s==AUXLANE){
```

```
                    int
r=(int)((1500*INITIALDENSITY*type1dense)/(30*100));
                              if(type1carnum>=r)
                                   goto over;
                              else
                          {
                                    car[enter].type=1;
                                   type1carnum++;
                              }
                     }
                        else
                        { if(type2carnum < type2carcome)
                        {
                            car[enter].type=2;
                                  type2carnum++;}
                                  else if(type1carnum < type1carcome)
                                  {
                                 car[enter].type=1;
                                        type1carnum++;}
                                  else
                                        goto over;
                                  }

if(c>4)
{
          if(((1500*INITIALDENSITY)/100)-carnum<=5)
car[enter].pos=myrandom(5);//4,3,2,1,0
else          if(((1500*INITIALDENSITY)/100)-carnum<=7          &&
((1500*INITIALDENSITY)/100)-carnum>5)

     car[enter].pos=myrandom(5-1)+1;//4,3,2,1

else          if(((1500*INITIALDENSITY)/100)-carnum<=10          &&
((1500*INITIALDENSITY)/100)-carnum>7)

     car[enter].pos=myrandom(4-1)+2;//4,3,2

else          if(((1500*INITIALDENSITY)/100)-carnum<=15          &&
((1500*INITIALDENSITY)/100)-carnum>10)

     car[enter].pos=myrandom(3-1)+3;//4,3

else          if(((1500*INITIALDENSITY)/100)-carnum<=30          &&
((1500*INITIALDENSITY)/100)-carnum>15)

     car[enter].pos=4;// only four
          totalcar++;
          car[enter].lane=s;
          car[enter].exit=0;
```

```
                c=car[enter].pos;
                car[enter].entry=car[enter].pos;
                road[c][s]=enter;
                car[enter].speed=5;
                initialcarnum++;
        car[i].ok=0;
             density=(float)((type1carnum+type2carnum)*100/1500);
            fprintf(carenterance,"%d.car(%d) enters to lane %d at
%d.second \n",enter,
           car[enter].type,s,t);
             enter++;
             break;
      }
           else
           {
        car[enter].pos=car[first].pos-1//max(0,(car[first].pos
      max(car[first].speed,
        safedistance)));
              car[enter].lane=s;
              car[enter].exit=0;
              car[enter].speed=car[first].speed;
              c=car[enter].pos;
              car[enter].entry=car[enter].pos;
              road[c][s]=enter;
              initialcarnum++;
              car[i].ok=0;
              density=(float)((type1carnum+type2carnum)*100/1500);
              fprintf(carenterance,"%d.car(%d) enters to lane %d at
      %d.second\n", enter,
        car[enter].type,s,t);
        enter++;
              break;
      }
     }else break;
     }else if(c==10)
          goto enterance;
      }
      }

      if(road[ONRAMPBEGIN][ONRAMPLANE]==0)  // loop for entering
      to onramp from the beginning of the onramp.
       {
         int r=0;
          for(r=ONRAMPBEGIN;r<=ONRAMPEND;r++)
          {
         //look for the first car and when entering adapt speed and position
      wrt to the first car.
              if(road[r][0]!=0)
        {
```

```
                int first=road[r][0];//farthest downstream car
                if(enter<=NUMBER){
                if(waitingcar*100/30 < ONRAMPDENSITY)//if onramp
density is exceeded.
                {
                 enter=initialcarnum+1;
                        car[enter].type=1;
                        waitingcar++;
               if(r>ONRAMPBEGIN+4)
        { //in order to avoid cars to exit from tenth.cell suddenly
                  car[enter].pos=ONRAMPBEGIN+4;
                  car[enter].lane=0;
                  car[enter].entry=car[enter].pos;
                  car[enter].exit=0;
                  r=car[enter].pos;
            road[r][0]=enter;
                  car[enter].speed=5;
                  initialcarnum++;
                  car[i].ok=0;
                  fprintf(onramp,"%d.car(%d) enters to onramp from
%d.cell  %d.second  \n",
            enter,car[enter].type,r,t);
                  enter++;
                  break;
        }
        else
   { //safedistance=car[first].pos;  //to imitate the adaptation of speed of
the entering car.
            car[enter].pos=car[first].pos-1;
     //max(0,(car[first].pos  max(car[first].speed,safedistance)));
            car[enter].lane=0;
            car[enter].entry=car[enter].pos;
            car[enter].exit=0;
            car[enter].speed=car[first].speed;
            r=car[enter].pos;
            road[r][0]=enter;
            initialcarnum++;
            car[i].ok=0;
            fprintf(onramp,"%d.car(%d) enters to onramp from %d.cell
%d.second \n", enter,
      car[enter].type,r,t);
            enter++;
            break;
   }
 }
 }
 }
}
}
```

```
int
allroad=0,type1allroad=0,type2allroad=0,type1allspeed=0,type2allspee
d=0;
 float allroaddensity=0.0,averagevelocity=0.0;
 for(l=4;l>0;l--)
{
         for(c=0;c<CELL;c++)
   {
        if(road[c][l]!=0)
         {
                  i=road[c][l];
                  if(car[i].exit==0)
             {
                        if(car[i].type==1)
                        {
                       type1allroad++;
                              type1allspeed+=car[i].speed;
                    }
                        else if(car[i].type==2)
                        {
                     type2allroad++;
                     type2allspeed+=car[i].speed;
                    }
                 }
                        allroad=type1allroad+type2allroad;
                        allroaddensity=float(allroad/(0.0075*1500));
                    //vehicles/km                          500*4/0.0075*1500=
averagevelocity=float(((type1allspeed+type2allspeed)*22.5)/allroad);
//5*22.5=112.5
                }
                        }
                    }
             fprintf(densitytime,"%d                %6f                %6f
\n",t,allroaddensity,averagevelocity);
             for(l=1;l<LANE;l++)
            {
                        for(c=0;c<CELL-10;c=c+10)
                        {
                                for(int f=c;f<c+10;f++)
                                {
                                        if(road[f][l]!=0)
                                        {
                                            if(f==1 && l==5)
                                                int y=0;
                                                i=road[f][l];
                                                add++;
                                                speed+=car[i].speed;
                                        }
                                }
```

```
                        dense[(c/10)+1][l]=dense[(c/10)+1][l]+(float)
add/10;

flow[(c/10)+1][l]=flow[(c/10)+1][l]+(float)speed/10;

avespeed[(c/10)+1][l]=avespeed[(c/10)+1][l]+(float)speed/add;
                        add=0;
                        speed=0;
    }
 }
 if (t%TIMEAVERAGE == 0)
{
        for (int f=1;f<=CELL/10;f++)
        {
                for (l=LANE-1;l>0;l--)
                {
                        dense[f][l] = (float)dense[f][l]/10;
                        flow[f][l] = (float)flow[f][l]/10;
                }
fprintf(detector, "%d    %f    %f      %f      %f      %f      %f      %f
%f                                                                    %f
%f\n",f,dense[f][5],flow[f][5],dense[f][4],flow[f][4],dense[f][3],flow[f][
3],dense[f][2],flow[f][2],dense[f][1],flow[f][1]);
        dense[f][1] = 0;
        flow[f][1] = 0;
        avespeed[f][5]=0;
        avespeed[f][4]=0;
        avespeed[f][3]=0;
        avespeed[f][2]=0;
        avespeed[f][1]=0;
        dense[f][2] = 0;
        flow[f][2] = 0;
        dense[f][3] = 0;
        flow[f][3] = 0;
        dense[f][4] = 0;
        flow[f][4] = 0;
        dense[f][5] = 0;
        flow[f][5] = 0;
    }
 }
 if(t>=500)
 {
         for(int y=0;y<CELL;y++)
         {
            trace[y][0]=0;
                trace[y][1]=0;
                trace[y][2]=0;
                trace[y][3]=0;
                trace[y][4]=0;
```

```
                trace[y][5]=0;
    }
            for(l=0;l<LANE;l++)
                {
                        for(int f=0,s=0;f<CELL;f=f++)
                        {
                                if(road[f][l]!=0)
                                {
                                        trace[s][l]=f;
                                        s=s++;
                                }
                        }
             }
            for(int f=0;f<CELL;f=f++)
            {
               if (trace[f][0]!=0 || trace[f][1] != 0 || trace[f][2] != 0 ||
               trace[f][3] != 0 || trace[f][4] != 0 || trace[f][5] != 0)
            fprintf(xtgraph,"%d %d %d %d %d %d
            %d\n",t,trace[f][5],trace[f][4],trace[f][3],trace[f][2],trace[f][1],tr
ace[f][0]);
            }


            for(c=CELL-1;c>=0;c--) //for each timestep,the photo of the
main road,auxiliary road
            fprintf(map,"%d %d %d %d %d %d %d
            %d.sec\n",c,road[c][5],road[c][4],road[c][3],road[c][2],road[c][1
],road[c][0],t);
 }
 }//iteration
fclose(carenterance);
fclose(Data);
fclose(map);
fclose(auxlane);
fclose(onramp);
fclose(offramp);
fclose (mainroad);
fclose(detector);
fclose(xtgraph);
fclose(densitytime);
fclose(control);
return 0;
}
```

REFERENCES

1. Schadschneider, A., "Statistical physics of traffic flow", *Physica A* , Vol. 285, No. 1, pp. 101-120(20),(2000).

2. Nagel, K. and A. Schrenkenberg, "A Cellular Automaton Model for Freeway Traffic", *J. Phys. I* France Vol. 2, Issue 12, pp 2221-2229 (1992).

3. Lighthill, M. J., and G. B. Whitham, *Proc. Roy. Soc. Lond. A* Vol. 229 No. 281 (1955).

4. Schrenkenberg, M., A. Schadschneider, K. Nagel, and N. Ito, "Discrete Stochastic Models for Traffic Flow", *Physical Review E*, Vol. 51, No. 4, pp. 2939,1995.

5. Priogine, I. and R. Herman,"Kinetic Theory of Vehicular Traffic", *New York:American Elsevier Publ., Co.*, Amsterdam,(1971).

6. Bellemans, T., B. De Schutter, and B. De Moor, "Models for traffic control", *Journal A.*, Vol.43., No.3-4, pp.13-22 (2002).

7. Herman, R., E. W. Montroll, R. B. Potts and R. W. Rothery, "Traffic Dynamics: Analysis of Stability in Car Following", *Operations Research*, Vol. 7, No., 1, pp. 186-106, (1959).

8. Payne, H. J., "Models of freeway traffic and control", *Mathematical Models of Public Systems Simulation Councils Proc. Ser.*, Vol.1, pp.51-61, (1971).

9. Kühne, R. D., "Macroscopic freeway model for dense traffic-stop-start waves and incident detection",Transportation and Traffic Theory", *VNU Science Press*, pp. 21-42, (1984).

10. Kerner, B. S., *"The physics of Traffic", emphSpringer, 2004.*

11. Helbing, D., and A. Greiner, "Modelling and simulation of multilane highway", *Phys.Rev.E* , Vol. 55, Issue 5, pp. 5498-5508 (1997).

12. May, A. D., and H. E. M. Keller, "Non-integer car-following models", *Highway Research Record*, No.199, pp. 19-32.(1967).

13. Benjaafar, S., K. Dooley, and W. Setyawan, "Cellular Automata for Traffic Flow Modeling", University of Minnesota, Minneapolis (1997).

14. Gazis, D. C., R. Herman, and R. B. Potts,"Car-following theory of steady-state traffic flow", *Operations Research 7*, Vol. 7, No. 4, pp. 499-505, (1959).

15. Gazis, D. C., "Traffic theory", *John Wiley Sons,Newyork*, (1973).

16. Herman, R., E. W. Montroll, R. B. Potts, and R. W. Rothery, "Traffic Dynamics: Analysis of Stability in Car Following", *Operation Research* , Vol.7, pp 86-106, (1959).

17. Kometani, E. and T. Sasaki, "On the stability of traffic flow", *Journal of Operations Research Japan* , Vol.2, pp. 1126 (1958).

18. Bando M. and K. Hasebe, "Dynamical model of traffic congestion and numerical simulation", *Phys. Rev. E*, Vol 51, No. 2, pp. 1035 - 1042, (1995).

19. Wiedemann, R., "Simulation des Straenverkehrsflusses", *Schriftenreihe des Instituts fur Verkehrswesen der Universitat Karlsruhe*,(1974).

20. Newell, G. F., "Memoirs on Highway Traffic Flow Theory in the 1950s", *Operations Research*, Vol. 50, No.1, pp. 173-178, (2002).

21. Nagel, K., "Particle hopping versus Fluid Dynamical Models for Traffic Flow", *Physical Review E*, Vol.53, Issue 5, pp.4655-4672, (1996).

22. Chowdhury, D., L. Santen, and A. Schadschneider, "Statistical Physics of Vehicular

Traffic and Some Related System", *Phys Rep.*, Vol.329, pp.199-329, (2000).

23. Hoefs, D. H., "Entwicklung einer Messmethode uber den Bewegungsablauf des Kolonnenverrkehrs" Universitat (TH) Karlsruhe, Germany, Vol. 19, No.4, pp 861-868, (1972).

24. Buckley, D. J., S. Yagar, "Capacity funnels near on-ramps", In: Buckley, D.J.(Ed.), *Transportation and Traffic Theory, Proceedings of the Sixth International Symposium on Transportation and Traffic Theory*, London, UK, pp. 105-123, (1974).

25. Nagel, K. and M. Paczuski, "Emergent traffic jams", *Phys. Rev. E*, Vol. 51, No. 4, pp. 2909-2918, (1995).

26. Nagel, K. and M. Paczuski, "Self organized criticality and 1/f noise in traffic", *Physical Review Letters E*, Vol.51, pp.2909, (1995).

27. Musha, T. and H. Higuchi, "Power-Law Fluctuation in Expressway Traffic Flow: Detrended Fluctuation Analysis", *Japan J. Appl.Phys.*, Vol. 15,pp. 1271, (1976).

28. Bak, P., "How Nature Works", *New York, Copernicus Press*,(1996).

29. Daganzo, C. F., M. J. Cassidy, and R. L. Bertini, "Causes and Effects of Phase Transitions in Highway Traffic", *Institute of Transportation Studies Research Report*, UCB-ITS-RR-97-8, (1997).

30. Fourrate, K. and M. Loulidi, "Disordered Cellular Automata Traffic Flow Models", *The Moroccan Statistical and Condensed Matter Society*, Vol.5, No.2, pp.151, (2004).

31. Herty, M. and A. Klar, "Modeling, Simulation, and optimization of traffic flow networks", *Sam J. Sc. Comput.*, Vol.25, No.3, pp.1066-1087, (2003).

32. Richards P. I., "Shock waves on the highway", *Operations Research*, Vol.4, No.1, pp. 42-51, (1956).

33. Waldeer,K. T., "Numerical Investigation of a MesoscopicVehicular Traffic Flow Model Based on a Stochastic Acceleration Process", *Transport Theory and Statistical Physics*, Vol.33, No.1, pp. 31-46, (2004).