Royal Institute of Technology

Degree Project in Engineering Physics

Numerical Analysis at the Department of Mathematics

Microscopic Models for Traffic Dynamics

Author:
Felix Abrahamsson

Supervisor:
Anders Szepessy

Examiner:
Mårten Olsson

May 15, 2016

Abstract

As the volume of vehicular traffic has been increasing ever since its advent in the early 20th century, traffic dynamics has become a popular topic of study amongst physicists and mathematicians. This paper aims to explain different ways of modeling and simulating traffic and traffic dynamics. In general, modeling of traffic can be divided into two types of models, microscopic and macroscopic. In this paper, primary focus is on microscopic models. Examples are shown on how to implement simulations of so called *car-following models* as well as models based on *cellular automata*. Certain problems and scenarios regarding traffic are studied as well. These include how to efficiently distribute a roadblock, how to set the green/red time periods of a traffic light in order to achieve a high traffic flow or a low vehicle density, and how traffic flow can be maximized with respect to density or other parameters. Results show that simulations using cellular automata models generally compute faster than simulations using car-following models. However, the results obtained from cellular automata models tend to be more difficult to interpret and apply to real-world scenarios. A high level of stochasticity in the cellular automata models was also found to be necessary for the models to give applicable results. Car-following models, on the other hand, were found to have the advantage of being more deterministic.

Contents

1 Introduction

Traffic and traffic dynamics has been a topic of study since the early 1930s [1]. Finding mathematical models for traffic has become increasingly necessary as the volume of vehicular traffic has been growing at a faster rate than the capacities of the road networks. Today, the time European drivers spend in traffic jams amounts to several days each year, and traffic jams may grow as long as 100 kilometers during holiday seasons. In Germany, the economic loss due to congested traffic has been estimated to be of the order of magnitude of 100 billion euros [1]. These issues, along with the the desire to understand seemingly hard-to-explain phenomena such as "phantom traffic jams" that occur without any observable cause, have stimulated research by many physicists in order to find mathematical models to describe traffic dynamics.

In general, traffic models can be divided into two categories - micromodels and macromodels. In micromodels vehicles are treated as discrete entities that can have different properties, whereas in macromodels all the vehicular information is lumped together into continuous variables such as vehicle density, velocity and vehicle flow that may depend on time and position on the road.

When studying ideal gases in thermodynamics it is not necessary to know the properties of each particle in the gas in order to make predictions about temperature and pressure. With this analogy in mind, one can see how macromodels are useful in studying collective phenomena, such as how traffic jams develop over time and how velocity and traffic flow changes. Micromodels, on the other hand, have the advantage of giving the ability to visualize directly how individual vehicles are affected by other vehicles, traffic lights, speed limits and so on. The obvious disadvantage of micromodels is computation time; once the number of vehicles grows too large it simply becomes time inefficient to simulate with the use of micromodels.

It is important to make the distinction between *traffic modeling* and *transportation planning* [2]. Traffic modeling assumes a time scale that does not exceed a few hours, and focuses on modeling the operations of the drivers, such as acceleration and lane changing. Transportation planning assumes a larger time scale, between hours and years. High level operations of the drivers, such as the choice of destination and the choice of route, belong to transportation planning and is not in the scope of traffic modeling.

This report will attempt to describe various traffic models, and focus specifically on micromodels and the implementation and simulation of such models.

1.1 Problem formulation

The main problems treated in this report are the following:

- Comparing different micromodels for traffic - which are most advantageous for simulation?

- Given a road with a traffic light, it can be assumed that at a certain value of the vehicle density traffic jams begin to grow indefinitely depending on the amount of time the traffic light is green/red. At this limiting density, is it possible to find a relationship between the density and the green/red time of the traffic light?

- Is it possible to determine how to most efficiently distribute the blocking of a road (during roadwork, for instance) in order to maximize traffic flow or velocity?
- Given certain fixed parameters, what vehicle density maximizes vehicle flow?
- Given a line of cars behind a traffic light, how fast is the line emptied once the light turns green depending on traffic model, distance to other vehicles, or other parameters?

2 Theoretical background

2.1 Conservation laws and macromodels

Macromodels mainly describe traffic using theory of fluid dynamics [2]. Often the road is described by a one-dimensional position coordinate, x, and multiple lanes are grouped together in the model. Among the commonly used parameters are the vehicle density $\rho(x,t)$ [vehicles/m], the local velocity $V(x,t)$ [m/s] and the vehicle flow $Q(x,t)$ [vehicles/s] that all depend on the position as well as the time, t. The hydrodynamic relation between these parameters can be formulated as

$$Q(x,t) = \rho(x,t)V(x,t). \tag{1}$$

If we let Δx be microscopically large enough so that the macroscopic assumptions about ρ still hold, but macroscopically small enough so that ρ is constant on the road section $[x, x+\Delta x]$ the number of vehicles, $n(t)$, on that road section can be calculated as

$$n(t) = \int_x^{x+\Delta x} \rho(x',t)dx' \approx \rho(x,t)\Delta x. \tag{2}$$

The assumption can be made that n only changes due to inflow, $Q_{\text{in}} = Q(x,t)$, or outflow, $Q_{\text{out}}(x+\Delta x,t)$ of vehicles in the road section. This can be formulated as

$$\frac{dn}{dt} = Q(x,t) - Q(x+\Delta x,t).$$

Equation (2) gives

$$\frac{\partial\rho}{\partial t} = \frac{1}{\Delta x}\frac{dn}{dt} = -\frac{Q(x+\Delta x,t)-Q(x,t)}{\Delta x} \to -\frac{\partial Q}{\partial x} = \{\text{eq. (1)}\} = -\frac{\partial(V\rho)}{\partial x}.$$

This leads to the continuity equation

$$\frac{\partial\rho}{\partial t} + \frac{\partial(V\rho)}{\partial x} = 0. \tag{3}$$

This equation governs the dynamics of what is called *first-order models*, where the flow is given as a static function of the density, $Q(\rho) = \rho(x,t)V(\rho(x,t))$ [1, 2]. One may for instance use the linear model $V(\rho) = 1-\rho$, where free stream velocity and the maximum density (where the vehicles are bumper to bumper) are both set to 1. This results in the nonlinear, partial differential equation

$$\frac{\partial\rho}{\partial t} + \frac{\partial\left(\rho(1-\rho)\right)}{\partial x} = 0.$$

Opposed to first-order models, *second-order models* assume a dynamical acceleration equation for the velocity, V, based on the material time derivative [2]:

$$\frac{dV(x,t)}{dt} = \left(\frac{\partial}{\partial t} + V(x,t)\frac{\partial}{\partial x}\right) V(x,t) = A\left(\rho(x,t), V(x,t)\right)$$

where A is the acceleration function. This acceleration model, coupled with the continuity equation (3) governs the second order models. One such example is Payne's model, in which the acceleration model is assumed to be [2]

$$\frac{\partial V}{\partial t} + V\frac{\partial V}{\partial x} = \frac{V_e(\rho) - V}{\tau} + \frac{V_e'(\rho)}{2\rho\tau}\frac{\partial \rho}{\partial x}, \tag{4}$$

where prime denotes derivative, $V_e(\rho)$ is the steady-state velocity, τ is the speed relaxation time, corresponding to the time it takes for the system to reach steady-state.

2.2 Car-following models

One type of microscopic model for traffic flow is car-following models. Car-following models describe individual vehicle actions based on the surrounding traffic, and thus do not describe free traffic flow [2]. Commonly used state parameters and variables are as seen in Fig. 1 below. Note that while it is possible to model multiple lanes and lane changing with car-following models, the car-following models in this report are restricted to model only single lanes.

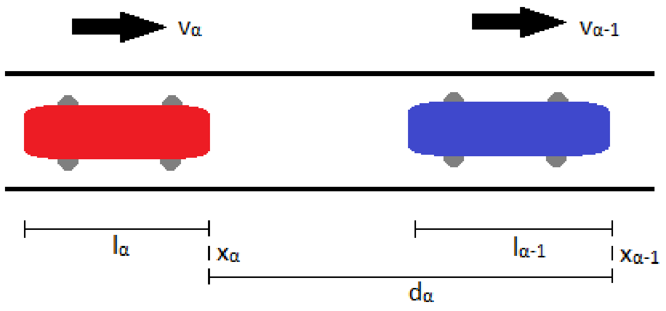


Figure 1: Each vehicle is labeled by an index α, which increases from the leading vehicle backwards on the road (upstream).

The position on the road, x, is usually taken to be the position of the front bumper of the vehicle. The vehicle lengths are denoted l, the velocites v, the distance between front bumpers d. The distance s_α between car α and the vehicle in front of it is therefore

$$s_\alpha = d_\alpha - l_{\alpha-1} = x_{\alpha-1} - x_\alpha - l_{\alpha-1}. \tag{5}$$

In the Newell model (and many other car-following models) the drivers are assumed to have a constant reaction time T [1, 2]. The velocity of vehicle α is then assumed to follow the law

$$v_\alpha(t+T) = v_{\text{opt}}(s_\alpha,\ v_\alpha,\ v_{\alpha-1}) \tag{6}$$

where $v_{\text{opt}}(s_\alpha,\ v_\alpha,\ v_{\alpha-1})$ is some specified "optimal velocity function" that depends on the different state variables. In the Newell model specifically, the function v_{opt} follows the relation

$$v_{\text{opt}}(s_\alpha(t)) = \min\left(v_0,\ \frac{s_\alpha(t)}{T}\right) \tag{7}$$

in which vehicle α keeps the constant velocity v_0 as long as the distance to the leading vehicle does not exceed v_0T. If the distance exceeds v_0T the vehicle keeps a velocity that ensures collisions do not happen. The Newell model for the function v_{opt} assumes an instant change of velocity (corresponding to infinite acceleration, which is an obvious shortcomming of the model), but the theory can be extended to include acceleration. One such extension is the Gipps' model, which attempts to define a "safety velocity", v_{safe}, by assuming the following [2]:

- Vehicles accelerate with constant acceleration a.
- Vehicles decelerate with constant deceleration b.
- The distance to the leading vehicle may not fall below a distance s_0.
- The safety velocity, v_{safe}, depends on the the velocity of the leader and the distance s_α to the leader.

The model can be summarized as

$$v_\alpha(t+T) = \min\big(v_\alpha(t) + aT,\ v_0,\ v_{\text{safe}}(v_{\alpha-1}(t), s_\alpha(t)\big). \tag{8}$$

An expression for v_{safe} can be derived by calculating the distance it takes for vehicles α and $\alpha-1$ to come to a full stop [2]. The leading vehicle requires the time $\Delta t_{\alpha-1} = \frac{v_{\alpha-1}(t)}{b}$ to stop, resulting in the required distance

$$\Delta x_{\alpha-1} = \int_0^{v_{\alpha-1}} \Delta t_{\alpha-1} dv'_{\alpha-1} = \int_0^{v_{\alpha-1}} \frac{v'_{\alpha-1}}{b} dv'_{\alpha-1} = \frac{v_{\alpha-1}^2}{2b}.$$

In order to find a similar expression for the stopping distance of vehicle α its reaction time needs to be taken into account. The stopping distance is therefore

$$\Delta x_\alpha = v_\alpha T + \frac{v_\alpha^2}{2b}.$$

Applying the safety restriction that for vehicle α the distance s_α must exceed s_0 by $\Delta x_\alpha - \Delta x_{\alpha-1}$, the condition on s_α can be formulated as:

$$s_\alpha \geq s_0 + v_\alpha T + \frac{v_\alpha^2}{2b} - \frac{v_{\alpha-1}^2}{2b}. \tag{9}$$

The highest possible velocity vehicle α is allowed to keep to avoid falling below the distance s_0 to the leading vehicle is given by equality in equation (9):

$$\begin{aligned} s_\alpha &= s_0 + v_\alpha T + \frac{v_\alpha^2}{2b} - \frac{v_{\alpha-1}^2}{2b} \Leftrightarrow \\ v_\alpha^2 + 2bT \cdot v_\alpha &= v_{\alpha-1}^2 + 2b(s_\alpha - s_0) \Leftrightarrow \\ (v_\alpha + bT)^2 - b^2T^2 &= v_{\alpha-1}^2 + 2b(s_\alpha - s_0) \Leftrightarrow \\ v_\alpha = v_{\text{safe}} &= -bT + \sqrt{b^2T^2 + v_{\alpha-1}^2 + 2b(s_\alpha - s_0)}. \end{aligned} \tag{10}$$

Equation (8) coupled with equation (10) completes the Gipps' model.

2.2.1 Link between microscopic and macroscopic models

The link between the microscopic models treated in Section 2.2 and macroscopic models can be derived approximately by the following reasoning [2]. Assume that the local velocity $V(x,t)$ can be taken to be

$$V(x,t) = V(x_\alpha(t),t) = v_\alpha(t).$$

Performing a Taylor expansion of equation (6), omitting terms of second order or higher and using the above expression results in

$$\begin{aligned} v_\alpha(t+T) &= V(x_\alpha + v_\alpha T, t+T) \\ &\approx V(x_\alpha,t) + \frac{\partial V(x,t)}{\partial x} v_\alpha T + \frac{\partial V(x,t)}{\partial t} T \\ &= V(x,t) + \left(V(x,t)\frac{\partial V(x,t)}{\partial x} + \frac{\partial V(x,t)}{\partial t} \right) T. \end{aligned} \tag{11}$$

Defining the density as (see Fig. 1)

$$\frac{1}{\rho} = d_\alpha = s_\alpha + l_{\alpha-1}$$

and making the assumption that the steady state velocity V_e takes the value of the optimal velocity function, so that

$$V_e(\rho) = v_{\text{opt}}(s_\alpha) = v_e(s).$$

By evaluating ρ at the midpoint, $x_\alpha + \frac{d_\alpha}{2}$, between two vehicles, a Taylor expansion results in:

$$\begin{aligned} v_{\text{opt}}\big(s_\alpha(t)\big) &= V_e\left(\rho\left(x + \frac{d_\alpha}{2}, t\right)\right) \\ &\approx V_e\big(\rho(x,t)\big) + V_e'(\rho)\frac{\partial \rho}{\partial x}\frac{d_\alpha}{2} \\ &= V_e(\rho) + \frac{V_e'(\rho)}{2\rho}\frac{\partial \rho}{\partial x}. \end{aligned} \tag{12}$$

Equation (6) states that expressions (11) and (12) are equal, resulting in

$$\frac{\partial V}{\partial t} + V\frac{\partial V}{\partial x} = \frac{V_e(\rho) - V}{T} + \frac{V_e'(\rho)}{2\rho T}\frac{\partial \rho}{\partial x}. \tag{13}$$

By identifying the reaction time, T, in the above equation as the speed relaxation time, τ, from Payne's model (equation (4)) we find that the microscopic models that are based on equation (6) are always approximately equal to Payne's model.

2.3 Cellular automata

Another way of modeling traffic microscopically is through cellular automata [1]. In cellular automata, space is discretized into a one- or two-dimensional lattice consisting of cells. Each cell is either empty or occupied by an object, and the entire cellular automaton (i.e. the lattice cells) evolves in discrete time steps Δt based on a set of rules that determine what happens to each cell based on its neighboring cells. The idea of cellular automata was first introduced in 1963 by Neumann and Ulam as a way of idealizing biological systems [3]. As the concept has been further applied to other physical systems, it has been found that cellular automata can be used to approximate many types of physical systems that can be described by discrete elements with local interactions governed by differential equations.

In cellular automata models for traffic dynamics, the road is discretized into an array of cells, each cell of length Δx (typically 2-3 car lengths) [2]. The array may be two-dimensional, simulating multiple lanes, or one-dimensional, simulating a single lane. A cell is either occupied or unoccupied by a vehicle, and vehicles move to cells further ahead based on specified rules. Depending on the model of choice, a single vehicle may occupy one or multiple cells. A visual representation of a section of the road in a cellular automata model can be seen in Fig. 2.

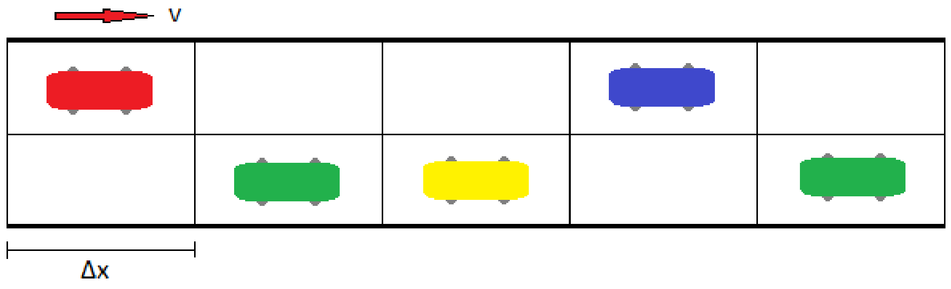


Figure 2: A two-lane road in a cellular automata model where each cell is Δx long. The velocity of the red car is v.

A simple model is the Nagel-Schreckenberg model, in which the road consists of a single lane and each vehicle occupies one cell [1]. A vehicle has the velocity $v^*(t) = v(t)\frac{\Delta x}{\Delta t}$, so that each time step Δt the vehicle moves v cells. Note that v^* has the dimension $\frac{\text{length}}{\text{time}}$ while v has no dimension. The velocity v is not allowed to exceed the value $v_{\max}$. Typical values for freeway traffic are $\Delta x = 7.5\text{m}$, $\Delta t = 1\text{s}$ and $v_{\max} = 5$, corresponding to a maximum velocity of 135 km/h [1]. Each vehicle has their position and velocity updated according to the following steps:

1. A vehicle has velocity v.

2. The velocity is accelerated to $v' = v + 1$, as long as $v' \leq v_{\max}$.

3. The velocity is decelerated to $v'' = d - 1$, where d is the number of cells between the vehicle and the next vehicle in the array, as long as $d \leq v'$.

4. With some probability p the velocity is decreased to $v''' = v'' - 1$, as long as $v''' \geq 0$.

5. The new velocity of the vehicle is $v = v'''$, and it moves v''' cells in the next time step.

The above steps can be summarized as

$$v_{\text{next}} = \max\big(0, \min(v_{\max}, d-1, v+1) - \chi(p)\big) \tag{14}$$

where $\chi(p) = 1$ with probability p, and 0 with probability $1-p$. The stochastic variable χ represents a tendency of the driver to dawdle and lose velocity. The model can be further built upon by letting the value of p depend on the velocity of the vehicle, for instance

$$p = \begin{cases} p', & v > 0 \\ p_0, & v = 0 \end{cases} \tag{15}$$

where $p_0 > p'$ corresponds to a slow-to-start tendency while standing still in traffic jams. The model can also be extended to include multiple lanes, though it introduces the problem of defining the distance d in a distinct way.

Cellular automata models like this one have the advantage of fast computation during simulation. They are particularly suitable for parallel computing and can be used to simulate a large number of vehicles simultaneously, which is a property that most other micromodels lack.

3 Implementation and simulation

For this project, implementation and simulation of different traffic models was done entirely using Python. The first model to be implemented was a cellular automaton, and later on a simulator for car-following models was built. Main focus was put on the cellular automata model, as it turned out to be the most useful one for studying the problems specified in Section 1.1.

3.1 The cellular automata model

The cellular automata model used for simulation is based on the Nagel-Screckenberg model (see equation (14)). The simulator is comprised of a main program and two classes named `Simulator` and `Object`. The main program serves the purpose of drawing a graphical user interface (GUI) to control and run the simulator. The `Simulator` class contains all the model-specific parameters and variables (which are passed from the main program) and performs all the operations necessary for the cellular automaton to function. The `Object` class is used to create objects that occupy the cells, which can be either a car or a roadblock.

The road is represented by a list containing another list for each lane. Thus, each cell can be reached by specifying coordinates (i, j), where i corresponds to the lane number, and j to the downstream position on the road. A cell is either empty or occupied by an Object.

Certain rules were added to the model that deviated from, or were not included in, the original Nagel-Schreckenberg model. They are as follows:

- For multiple lanes, the value of d is taken as the number of empty cells between a vehicle and the nearest laterally blocked section of the road. For instance, if there is a vehicle at coordinates $(0, j)$ on a two-dimensional road, and the next two vehicles are at positions $(0, j+3)$ and $(1, j+3)$ the number of cells d is 2. If the next two vehicles instead are at positions $(0, j+3)$ and $(1, j+2)$, the number of cells d is not restricted by those vehicles. Note that this definition allows the vehicle to go past a diagonally occupied road. This definition requires the assumption that there is enough extra space in the cells to allow passing diagonally.

- Vehicles can change lanes with no penalty in velocity or time. Combining this with the above bullet point, an example can be made from Fig. 2: if we assume that the red vehicle wants to move 4 cells, and all other vehicles want to stand still, the red vehicle will end up in the cell in front of the blue vehicle.

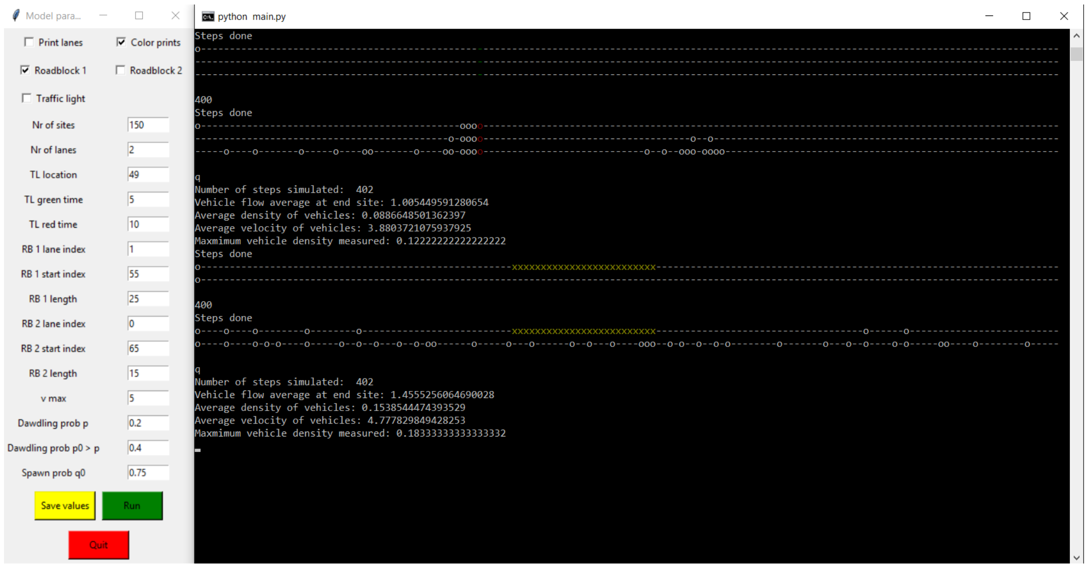
- Vehicles prioritize the rightmost lane when updating their position, so that if a vehicle in the leftmost lane has velocity $v = 5$, and the cell 5 cells downstream in the rightmost lane is unoccupied, the vehicle moves to that cell. If that cell is occupied, but the cell to the left of it is not, the vehicle would move to that cell instead.

- Vehicles enter the first cell in the array (at coordinates $(i, 0)$) based on probability. If cell $(i, 0)$ is empty, there is a probability q_0 that a car is placed in the cell. The velocity of the newly placed car is $v = d - 1$, with d calculated in the same way as formulated above. The value of q_0 therefore corresponds to a value of the vehicle density for free traffic.

- Vehicles exiting the final cells of the array are removed entirely from the simulator.

- The velocity and the position of each vehicle is not updated in parallel, but sequentially. In one time step the updates occur starting with the rightmost lane at the last index of the array, and continue with the next lane before going upstream of the road. This was done because there is no definitive way to apply parallel updating for a multilane model that does not result in multiple vehicles occasionally choosing to move to the same cell.

For visual representation of the array, simple console output is used with color coding. For colored console output, the `colorama` module is used. The current state of the cellular automaton is printed at first, and then the user specifies a number of time steps for the simulator to run. After the simulator has finished the steps, the new state is printed. The simulator can be seen in Fig. 3, with the console outputs to the right. The color and character codings seen in the console outputs are summarized in Table 1.



Model para...

- [ ] Print lanes
- [✓] Color prints
- [✓] Roadblock 1
- [ ] Roadblock 2
- [ ] Traffic light

Nr of sites	150
Nr of lanes	2
TL location	49
TL green time	5
TL red time	10
RB 1 lane index	1
RB 1 start index	55
RB 1 length	25
RB 2 lane index	0
RB 2 start index	65
RB 2 length	15
v max	5
Dawdling prob p	0.2
Dawdling prob p0 > p	0.4
Spawn prob q0	0.75

python main.py

```
Steps done
o-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------

400
Steps done
o---------------------------------------------oooo----------------------------------------------------------------------------------------------------
--------------------------------------------o-oooo------------------------------------o--o------------------------------------------------------------
-----o----o-------o-----o----oo-------o----oo-oooo----------------------------o--o--ooo-oooo----------------------------------------------------------

q
Number of steps simulated:  402
Vehicle flow average at end site: 1.005449591280654
Average density of vehicles: 0.0886648501362397
Average velocity of vehicles: 3.8803721075937925
Maxmimum vehicle density measured: 0.12222222222222222
Steps done
o------------------------------------------------------xxxxxxxxxxxxxxxxxxxxxxxxx----------------------------------------------------------------------
o-----------------------------------------------------------------------------------------------------------------------------------------------------

400
Steps done
o----o----o--------o--------o--------------------------xxxxxxxxxxxxxxxxxxxxxxxxx------------------------------------o------o--------------------------
o----o----o-o-o----o-----o--o--o---o--o-oo------o-----o---o------o--o---o----ooo--o-o--o--o-o--------o-------o---o--o----o-o-----oo----o--------o-----

q
Number of steps simulated:  402
Vehicle flow average at end site: 1.4555256064690028
Average density of vehicles: 0.1538544474393529
Average velocity of vehicles: 4.777829849428253
Maxmimum vehicle density measured: 0.18333333333333332
```

Figure 3: The cellular automata simulator. The GUI used to change model parameters can be seen to the left. A part of the visual representation of the cellular automaton from two different simulations can be seen as output in the console to the right, where 400 time steps have been run on each simulation. Console outputs containing simulation statistics can be seen printed after each finished simulation.

Entity	Car	Empty	Roadblock	Green light	Red light
Sign	o	-	x	o or -	o or -

Table 1: Summary of color and character codings for the console output in Fig. 3. Note that the colors of the first two entries in the table are inverted.

A way to simulate a traffic light was also incorporated into the model. A location for the traffic light, along with the amount of steps for the green and red periods of the traffic light, is specified in the `Simulator` class. Note that the traffic light in this model has no intermediate (yellow) period between red and green. The traffic light does not occupy cells in the array, but simply exists at a certain location allowing vehicles to occupy the cells if desired. If the traffic light is red, a vehicle has its value of d restricted by the traffic light in the same way as if there was another vehicle in the position of the traffic light. If the traffic light is green it has no effect on the simulation steps.

A GUI was also built to make changing model parameters and variables easier during simulation. The parameters and variables are then passed to the `Simulator` class when the user chooses to run the simulation through the GUI. The GUI can be seen to the left in Fig. 3.

The simulator was also made to calculate certain statistics for each simulation, such as the average vehicle density and velocity, the maximum measured density and the vehicle flow at the final cells of the array. These statistics could then be used to answer the questions specified in Section 1.1.

3.2 The car-following simulator

A simulator environment was also built for car-following models based on the model in equation (6) on a one-lane road. The simulator contains a graphical representation of the vehicles moving on the road, using the `Turtle` graphics module included in the standard Python 3 distribution. Similarly to the cellular automata simulator, this simulator consists of a main program and two classes named `Simulator` and `Object`. The main program has the task of creating a `Simulator` object, asking the user to input a number of steps for the simulator to run and then running the simulator. The `Simulator` contains the car-following models (i.e. the v_{opt} functions and the model parameters) and performs all the necessary operations for the models to function. The `Object` class is used to create vehicle and traffic light objects.

In the `Simulator` class, objects are stored in an array that is sorted with respect to the vehicles' positions. Since the road only contains one lane, the vehicles only have one position parameter, x. With the array constantly kept sorted this way, a vehicle only has to look at the next vehicle in the array to find its leading vehicle. Each object in the array is an instance of the `Object` class and has some attributes such as position and velocity regardless of whether the object is a vehicle or a traffic light. However, if the object is a vehicle it has other attributes and methods that the traffic light does not have, and vice versa for the traffic light. Each object has its own instance of the `Turtle` class to be used for graphical visualization of the object. All the methods to draw and control the graphics for each object are contained in the `Object` class as well. The graphical visualization of the simulator can be seen in Fig. 4.

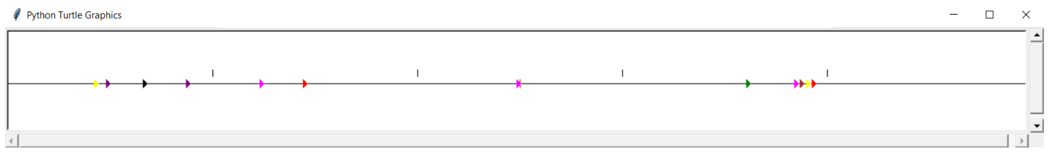


Figure 4: The car-following simulator. The road is represented by a one-dimensional line, and each vehicle is represented by an arrow with a random color. The vertical line in the middle of the road is a traffic light, and the red color indicates that the traffic light is red at the moment.

All the models in the simulator assume a constant reaction time T (see Section 2.2). The reaction time T is also the length of a simulation step, during which the positions and velocities are updated T/dt times, with dt being some fraction of T. Commonly used values for the simulations were $T = 1.0$ s and $dt = T/2000 = 0.0005$ s. In one step dt, each vehicle calculates its optimal velocity to be assumed at time $T + dt$ from the current time, t. This optimal velocity is then put in a queue named `next_velocities` (note: datastructure). After this the vehicle's next position is calculated as `new_position = current_position + new_velocity*dt`. The new positions and velocities are not applied until every vehicle in the array has performed their calculations, in order to achieve a form of simultaneous update. This way, each vehicle calculates their v_{opt} based on the same state of the traffic environment. This delayed update is also done so that sorting of the array does not change the order of the vehicles during their velocity calculations, since leader order can change between calculations.

Once every vehicle in the array has performed their calculations, each vehicle takes the next velocity from the `next_velocities` queue and assumes it as its velocity at time $t + dt$. The calculated next position is also applied as the vehicle's position for time $t + dt$. After that, the time step dt is concluded. The structure can be summarized as follows:

1. Each vehicle calculates its next velocity (for time $t + dt + T$) and position (for time $t + dt$).

2. Each vehicle applies its new velocity and position for time $t + dt$.

3. The vehicle array is sorted with respect to current positions.

In the `Simulator` class, a method named `v_opt` contains the different v_{opt} for each car-following model. In the implemented version of the simulator, the method contains the Gipps and the Newell models. The Newell model was modified to include a safety-distance s_0, defined as

$$s_0 = 2.0 \cdot v_\alpha + 2.0/(1 + v_\alpha),$$

which corresponds to the minimum desired distance to other vehicles. This definition is made to approximate the three-second rule (meaning that a vehicle at distance s_0 from its leader takes 3 seconds to reach the leaders current position, given its current velocity), while keeping a minimum value of 2 m at $v_\alpha \to 0$. For a vehicle with no leader, s is set to a large value (approaching infinity), in order to make certain types of simulations work. The modified Newell model can be summarized as

$$v_{\text{opt}} \ = \max\big(0, \min(v_0, (s - s_0)/T)\big).$$

The Gipps' model used in the simulator is exactly as specified in equations (8) and (10), with $a = 1.5$ m/s^2, $b = 1.0$ m/s^2 and $s_0 = 3$ m. Adding a new model was designed to be a simple matter, since all the state parameters s, v_α and $v_{\alpha-1}$ (see Section 2.2) are already calculated in the method.

As mentioned above, the simulator allows for simulation of a traffic light as well. The traffic light object is stored both in the vehicles' array and a specific array for traffic lights. The reason for storing it in its own designated array is to achieve a higher computation speed when updating the traffic light's state, and to allow for multiple traffic lights. The reason for storing it in the vehicles' array is so that it can be easily found when a vehicle is looking for the next object in front of it. The traffic light can be set to be green for a certain (integer) number of time steps T, and red for another number of time steps T. The traffic light has a "safety distance" parameter, which is included in the calculation of s in the `v_opt` function. In equation (5), this parameter functions as a substitute for $l_{\alpha-1}$. This safety distance can be regarded as the distance between the traffic light and its stop line. A red traffic light is treated by other vehicles as a still-standing vehicle when calculating v_{opt} as a function of a leading vehicle. A green traffic light is ignored entirely by all vehicles.

The simulator has three different modes. In all of the modes, the road has a finite length of `max_dist` = 5000.0 m. The first mode uses periodic boundary conditions, which means that vehicles exiting the road at position $x >$ `max_dist` return to the start of the road. In the second mode, vehicles exiting the road are removed from the program. New vehicles also have a probability of entering the road at $x = 0$ each time step T, with the probability being dependent on the value of T. The final mode is used to count how long it takes for a line of vehicles behind a red traffic light to pass once the light turns green. This mode sets up a specified amount of vehicles to drive and stop behind a red light. It then starts a timer once the light turns green, that stops once the last vehicle in the line has passed the traffic light.

3.3 Using the simulators to study problems in Section 1.1

The difference in efficiency between car-following models and cellular automata models was tested partially through time-taking of simulations, with different conditions. Other discussion regarding efficiency can be found in the Section 5.

3.3.1 Vehicle density vs. green time of traffic light

The cellular automata model was used to study the second bullet point in Section 1.1. First, the following parameters were fixed:

Nr of cells per lane	=	100
Nr of lanes	=	2
Location of traffic light	=	50:th cell
Traffic light red time	=	10 s
`v_max`	=	5
`p`	=	0.2
`p_0`	=	0.4

With the criteria specified above, simulations were run at 100,000 steps each. One such simulation would on average take about 70 seconds during congested traffic. The green time period of the traffic light was incremented by 1 s each simulation, for values between 1 and 16 s. For each value of the green time, the parameter `q_0` was increased until a critical value was found that caused traffic jams to grow indefinitely.

The criteria for an indefinitely growing traffic jam was defined to be when the average density over an entire simulation exceeded 0.4 vehicles/cell. Note that with the traffic light at the 50:th cell, if all cells behind the traffic light were occupied and all cells ahead of it were not the density would be 0.5 vehicles/cell. This criteria could have been defined with a different threshold than 0.4, or defined in an entirely different way. The reason for defining the criteria this way was because there was a relatively clear breakpoint where the average density would increase sharply with increasing `q_0`. A plot of how the average density changed with `q_0` can be seen in Fig. 5 in the *Results* section, during simulations of 30,000 steps each with the green time constantly at 10 s.

Once the majority of simulations (usually out of 9) resulted in the average density exceeding 0.4, the critical value of `q_0` was assumed to have been found.

3.3.2 Vehicle flow vs. vehicle density

The fourth bullet point regarding maximizing vehicle flow was tested using the cellular automata model, with the traffic light deactivated. The following parameters were fixed:

Nr of cells per lane	=	100
Nr of lanes	=	2
`p`	=	0.2
`p_0`	=	0.4

Given this, the parameter `q_0` was changed from 0.1 to 1.0, with increments of 0.1, and 50000 simulation steps were run. The value of the average vehicle flow was noted for each value of `q_0`. This was done with the parameter `v_max` set to values 5, 2 and 1 cells. The number of simulation steps was chosen by studying the variance of the average vehicle flow depending on the number of steps run.

3.3.3 Distribution of a roadblock

Distribution of roadblocking was tested by keeping the same parameters fixed as in Sec. 3.3.2, as well as the following:

Roadblock lane	=	rightmost lane
Roadblock start cell	=	50
`q_0`	=	0.7
`v_max`	=	5

Given this, the length of the roadblock was incremented by 1 cell at a time from 1 to 20, and 50000 simulation steps were run. The value of the average vehicle density was noted for each simulation. The value 0.7 for `q_0` was chosen because it was enough to give rise to traffic congestion at roadblock lengths around the midpoint of 10 cells.

3.3.4 A line of cars behind a traffic light

The final bullet point regarding a line of cars behind a traffic light was tested with the use of the car-following simulator (using the third mode mentioned in Section 3.2). First, the value $dt = T/2000$ was chosen as a fine enough mesh through testing of the convergence of results. Varying the number of vehicles in the line as well as the reaction time T, the amount of time it took for the line to pass the traffic light was noted. This was done for both the Newell and the Gipps' model.

4 Results

4.1 Simulation times

Results regarding simulation times for the car following models can be seen in Table 2 below, with $T = 1.0$ s. Similar results using the cellular automata model can be seen in Table 3 below. Note that since vehicles were initially put at random positions, the simulation times had some small variance and the results listed in the table were not completely consistent.

Model	dt	Nr. of vehicles	Graphics	Nr. of steps	Time taken
Gipps'	T/50	10	Deactivated	1000	5.18 s
Newell's	T/50	10	Deactivated	1000	4.92 s
Gipps'	T/100	10	Deactivated	1000	10.63 s
Gipps'	T/2000	10	Deactivated	1000	210.75 s
Gipps'	T/50	20	Deactivated	1000	10.49 s
Gipps'	T/50	10	Activated	1000	36.03 s
Newell's	T/50	10	Activated	1000	36.22 s

Table 2: Table over simulation times using the car-following simulator with different conditions applied. Note that all simulations used periodic boundary conditions and included a traffic light.

Traffic light	Green/red time	Avg. vehicles	Steps	Time taken
Not used	-	14.66	10,000	2.835 s
At cell 50	5/5 s	17.44	10,000	3.023 s
At cell 50	5/10 s	73.20	10,000	6.626 s

Table 3: Table over simulation times using the cellular automata model. The *Avg. vehicles* column specifies how many vehicles there were in the array on average. All simulations used 2 lanes with 100 cells per lane. Note that the final simulation in the table resulted in traffic congestion.

4.2 Probability `q_0` vs. traffic light green time

A plot of how the average vehicle density changed with `q_0` can be seen in Fig. 5. Simulations were run at 30,000 steps each, with the traffic light green time at 10 seconds in each simulation. Note the steep incline around `q_0` = 0.6, indicating the point where indefinitely growing traffic jams start to occur.

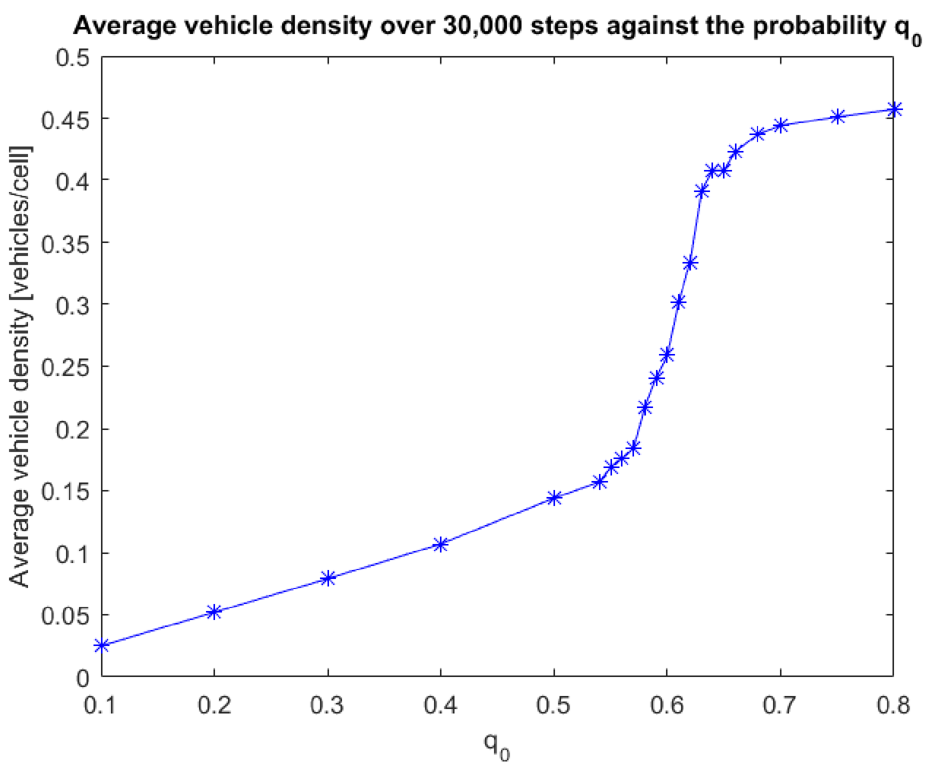


Figure 5: The average density over simulations of 30,000 steps each, with different values of the probability `q_0`.

A plot of the limiting probability `q_0` against the green time period of the traffic light can be seen in Fig. 6, and the data points can be found in Table 4.

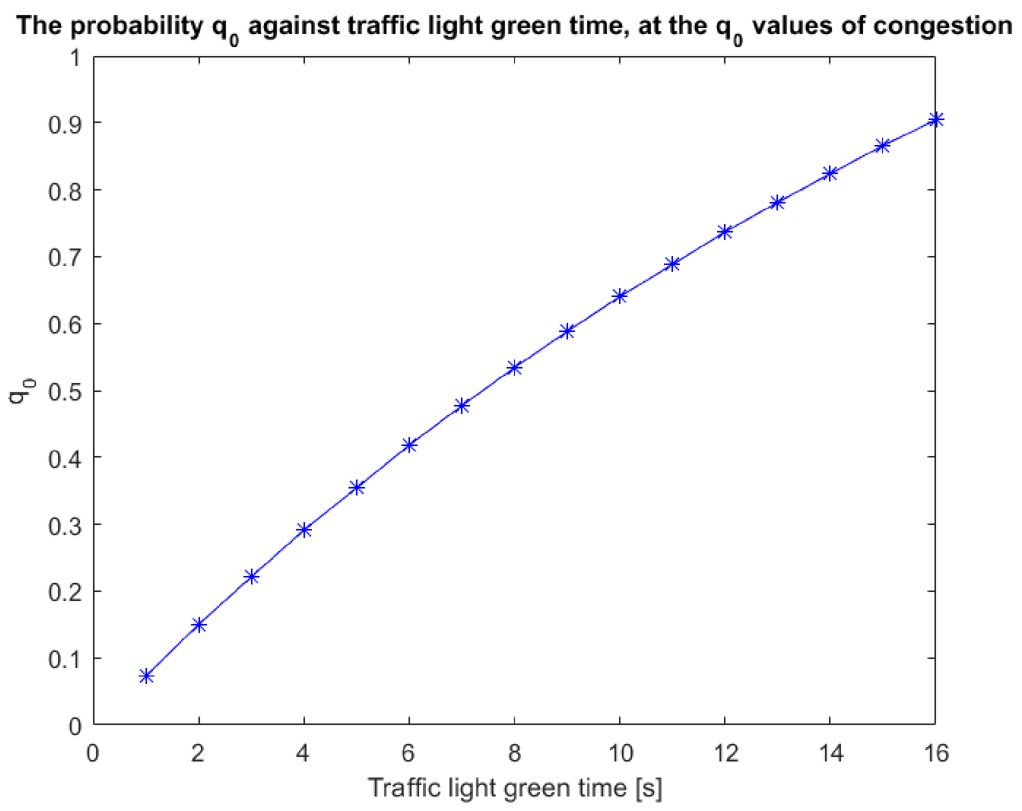


Figure 6: Plot of the probability q_0 against the amount of time the traffic light was set to be green, at the values of q_0 where they cause congestion.

Green time	1	2	3	4	5	6	7	8
q_0	0.073	0.150	0.221	0.291	0.354	0.418	0.477	0.534
	9	10	11	12	13	14	15	16
	0.588	0.640	0.688	0.737	0.781	0.824	0.866	0.904

Table 4: The data points corresponding to Fig. 6

4.3 Average vehicle flow vs. `q_0`

Plots over the average vehicle flow depending on the value of `q_0` can be seen in Figures 7, 8 and 9, with `v_max` set to the values 5, 2 and 1, respectively. The data points can be found in Table 5.

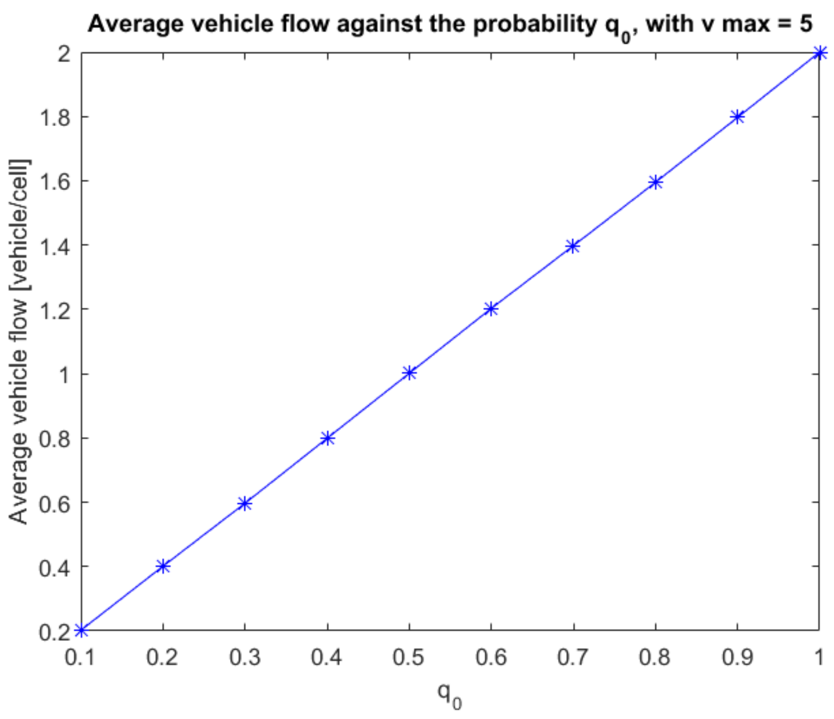


Figure 7: Plot over the average vehicle flow against the probability `q_0`, with `v_max` $= 5$

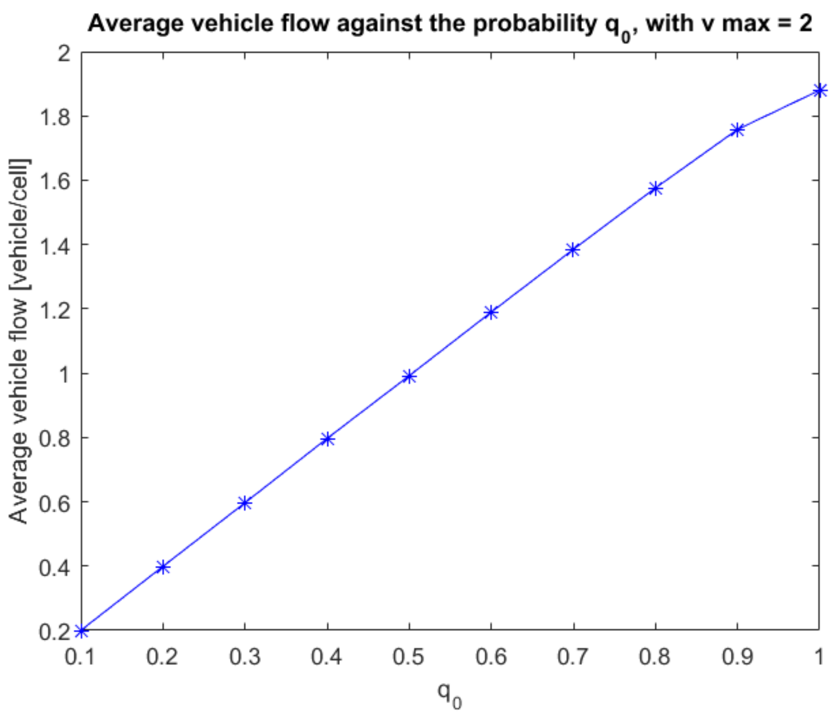


Figure 8: Plot over the average vehicle flow against the probability `q_0`, with `v_max` $= 2$

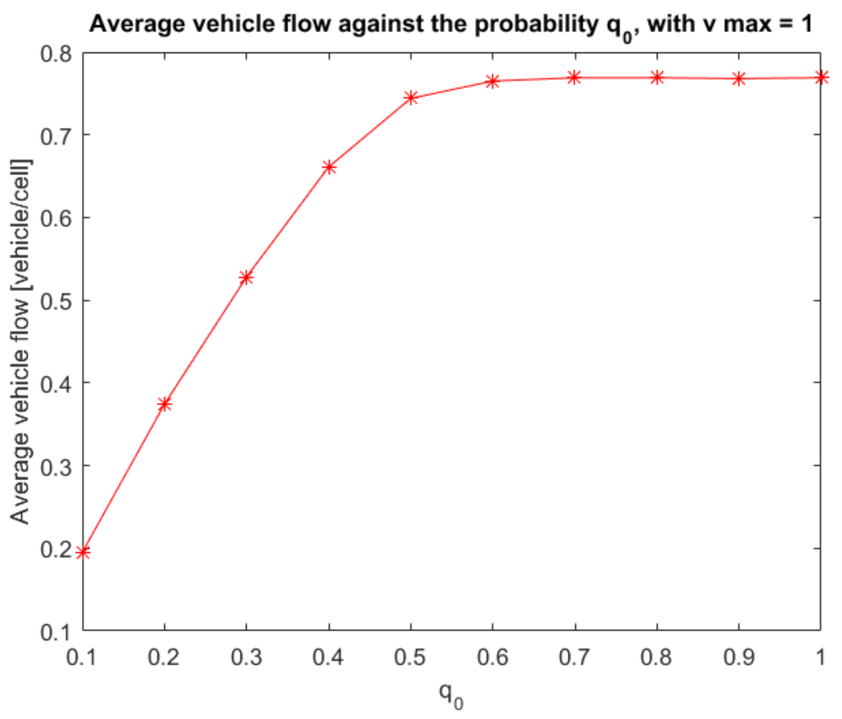


Figure 9: Plot over the average vehicle flow against the probability `q_0`, with `v_max` = 1

q_0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Avg.	0.202	0.401	0.597	0.799	1.001	1.203	1.398	1.595	1.797	2.000
flow	0.200	0.399	0.597	0.797	0.992	1.190	1.385	1.576	1.758	1.880
	0.195	0.374	0.528	0.661	0.744	0.765	0.769	0.769	0.768	0.769

Table 5: The data points corresponding to Figures 7 (top line in the *Avg. flow* row), 8 (the middle line) and 9 (the bottom line).

4.4 Distribution of a roadblock

Simulation of the distribution of a roadblock gave varied results. With the roadblock between lengths 1 and ∼5, the average vehicle density tended to settle around 0.149 vehicles per cell. Between 5 and 13 cells, the density would increase, but also fluctuate greatly. Five simulations with the roadblock at length 8 gave vehicle densities between 0.190 and 0.386 cars/cell. With the roadblock at lengths 13 to 20, the increase in vehicle density would begin to slow down, and settle somewhere between 0.40 and 0.43, with less fluctuations in the results. Simulations with longer roadblocks did not seem to increase the average density.

4.5 A line of cars behind a traffic light

Simulation of the time it took for a line of cars to pass a traffic light gave the results summarized in Tables 6 and 7. Figures 10, 11, 12 and 13 show plots over the data points.

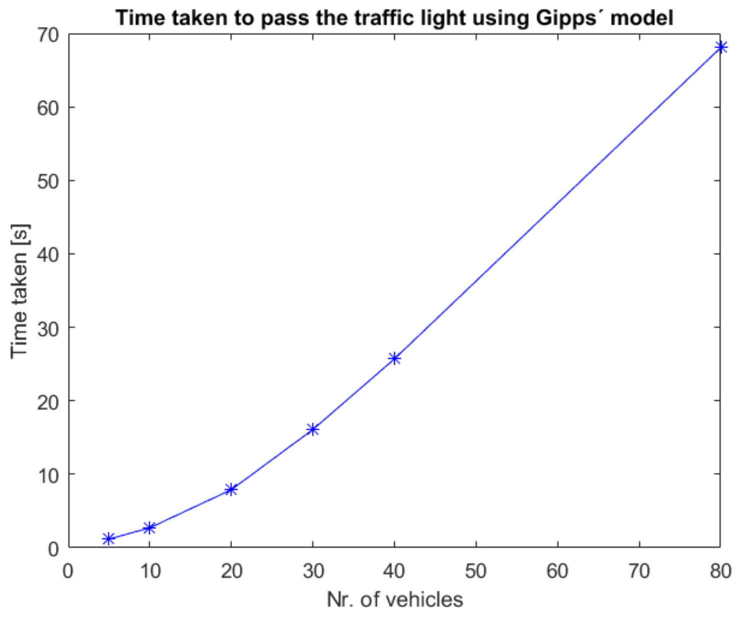


Figure 10: The amount of time taken for a line of cars to pass a traffic light depending on the number of vehicles, using the Gipps model.

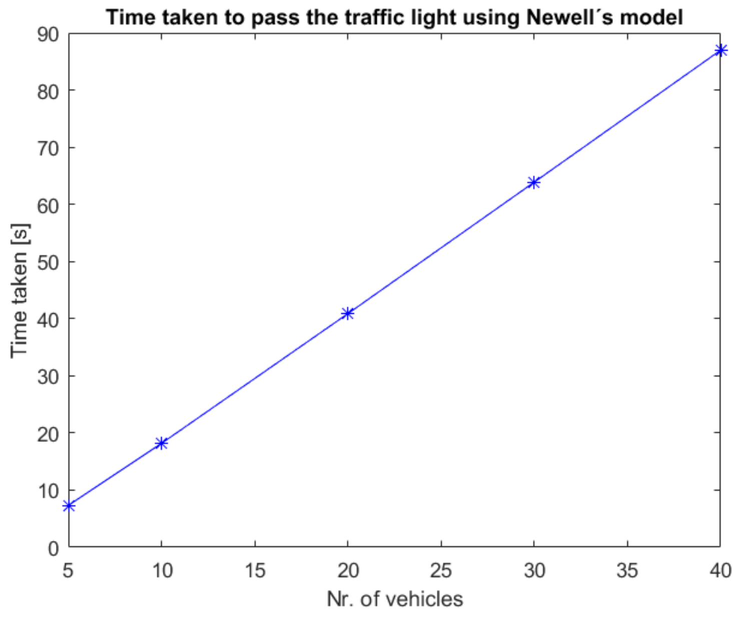


Figure 11: The amount of time taken for a line of cars to pass a traffic light depending on the number of vehicles, using the Newell model.

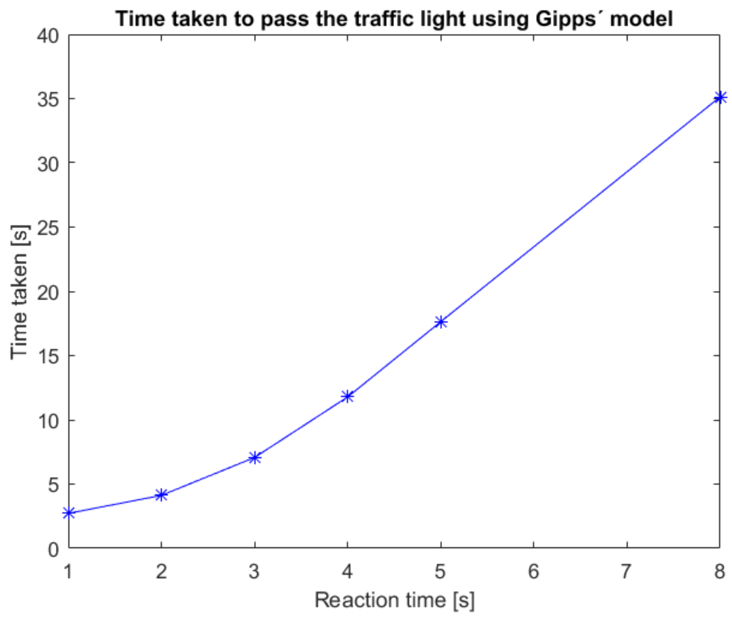


Figure 12: The amount of time taken for a line of cars to pass a traffic light depending on the reaction time, using the Gipps model.

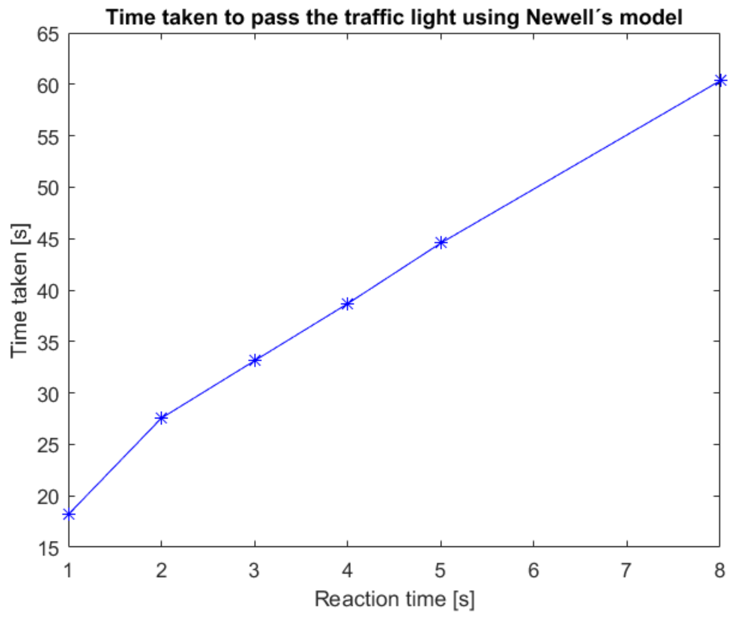


Figure 13: The amount of time taken for a line of cars to pass a traffic light depending on the reaction time, using the Newell model.

Nr of vehicles	5	10	20	30	40	80
Gipps:	1.19 s	2.72 s	7.91 s	16.10 s	25.71 s	68.06 s
Newell:	7.33 s	18.18 s	40.88 s	63.89 s	86.98 s	

Table 6: The top row shows the number of vehicles in the line behind the traffic light, the bottom two rows show results for the time it took for the lines to pass, for both the Newell and the Gipps model.

T	1 s	2 s	3 s	4 s	5 s	8 s
Gipps:	2.72 s	4.12 s	7.07 s	11.78 s	17.64 s	35.14 s
Newell:	18.18 s	27.55 s	33.13 s	38.68 s	44.57 s	60.35 s

Table 7: The top row shows reaction time of the drivers in the line, the bottom two rows show results for the time it took for the lines to pass the traffic light, for both the Newell and the Gipps model.

It should be noted that removing the introduced safety-distance s_0 from the Newell model reduced the passing times to around half of the listed values. It did, however, not change the characteristics of the curves.

5 Discussion and conclusions

5.1 Model and simulator efficiency

With regards to computation speed, the car-following simulator performed relatively slowly compared to the cellular automata simulator. Since the two simulators use very different models an exact comparison cannot be made, though it may be said that with a similar amount of vehicles and over similar time and length scales the cellular automata simulator computes faster. The mesh fineness (i.e the value of dt) in the car-following simulator turned out to be of more critical importance than expected. The results converged relatively slowly, and to achieve good convergence it was necessary to use values at, or smaller than, $dt = T/2000$. As seen in Table 2, this meant long computation times. The computation times seemed to increase linearly with increasing mesh fineness, as well as with the number of vehicles in the model, as can be expected since each car is only dependent on the one vehicle in front of it. Another conclusion that can be drawn from the results in Table 2 is that the graphics module introduced a considerable increase in computation time. With the graphics module activated the simulator performed roughly 7 times more slowly, which was not feasible when a fine mesh was required. A positive quality of the simulator with regards to computation speed though, is that the choice of v_{opt} model between Newell's and Gipps' did not affect the performance significantly, despite their difference in complexity.

Many improvements can be made to the simulator. For instance, instead of using an array to store the vehicles in, that constantly needs sorting, a linked-list structure can be used instead. With such a structure, each vehicle only needs to iteratively look at the next vehicle in the list to see if it is ahead of it during sorting. Another method would have been to use a *bubble-sort* algorithm in conjunction with the array structure, considering that the conditions are ideal for bubble-sort. A different choice of graphics module besides the `Turtle` module is likely to make the simulator run much faster, too. Since the graphics were updated sequentially, it is not surprising that it added computation time. However, for the purpose of performing calculations and answering the problems in Sec. 1.1, graphic visualization was not a necessity and was constantly deactivated during simulations.

The cellular automata simulator had the advantage being able to perform a large amount of simulation steps in a relatively short time. Table 3 also shows that the computation time did not increase linearly with the number of vehicles, but instead less than linearly. This is most likely because the program checked every cell in the array during each time step regardless of whether or not it was occupied by a vehicle, and thus did not perform many more operations if the cell happened to be occupied. The cellular automata simulator did, however, have the drawback of being highly probabilistic. Because of this, a very large amount of steps had to be run in order to achieve a small variance in the results. It was also more difficult to compare or apply the results to reality, due to the more diffuse connections between parameters and real world values. For instance, while there is a connection between the probability `q_0` and the density of free traffic, the exact relationship is not clear.

The cellular automata simulator was particularly convenient to use in simulating different traffic scenarios, such as introducing a roadblock, multiple lanes and lane-changing, traffic lights, on- and off-ramps etc. This was partially because of the breakdown of space into discrete cells.

It would be possible to introduce a model into the cellular automata simulator that allowed a vehicle to occupy multiple cells, thus reducing the corresponding real-value (meters) length of the cells. This would make the model approach the continuous car-following case more.

An improvement that can be made to both simulators would be to use a different programming language. More low-level languages would perform faster, and since computation time turned out to be an important factor this would be an improvement. Using parallel programming for the updating of vehicle states would also make both simulators perform faster.

5.2 Traffic light green time and vehicle density

Figure 6 shows an almost linear relationship between `q_0` and the traffic light green time at low values for the green time. At higher green time periods, the value of `q_0` begins to approach 1, as expected. However, the convergence seems very slow, which might be because of the choice of criteria for traffic congestion. As mentioned in Section 5.1, it is difficult to make comparisons between `q_0` and real world parameters, which makes it difficult to draw further conclusions from Figure 6. Because of the large amount of time it took to perform simulations with these conditions, it was not possible within the time frame of this project to do it with different conditions. Further simulations with different conditions would have allowed for more comparisons and possibly more conclusive results.

Figure 5 shows an interesting feature. The average density increases linearly with `q_0` at low values, corresponding to free flowing traffic. At values 0.54 <`q_0`< 0.65, the curve steepens significantly. It is at this point that traffic congestion begins to occur. At values above 0.65, the value of the density saturates, because of the limiting length of the array. It can therefore be assumed that around the value 0.65 and above, the traffic jam would increase indefinitely if the array length was not bounded.

5.3 Average flow and vehicle density

Figure 7 shows a linear relationship between the average vehicle flow and the probability `q_0`. This is because the traffic is on average free for all values of `q_0`. In Figure 9, however, the relationship starts out as linear but begins to approach a constant value for the average flow around `q_0` ≈ 0.5. At higher values of `q_0`, traffic is completely congested. The reason the flow does not increase further is likely because at `q_0`>0.5, on average almost every cell is occupied at all times, and each vehicle moves with the same velocity. It is difficult to determine where the maximum flow occurs, though it seems to occur after traffic congestion happens. It is possible that the maximum would occur before congestion for other values of `v_max`, since at complete congestion vehicles are unable to reach values of `v` greater than 1 cell. In the simulations that resulted in Figure 9, the number of cells `v_max` was set to 1 and `v` was therefore not affected by this restriction.

In Figure 8, the relationship remains linear almost up to `q_0` $= 1.0$, but the increase in flow begins to slow down as `q_0` approaches 1. The maximum still occurs at `q_0` $= 1.0$. Because this is the limiting value for `q_0`, and because `q_0` is the only parameter in the model that controls the free flowing traffic density, it is not possible to investigate whether a higher density would result in the maximum occurring before traffic congestion. This is a limitation of this implementation of the cellular automata model. The value `q_0` $= 1.0$ does not correspond to every cell being occupied at all times. With `v_max` > 1, each vehicle entering the array will move `v_max` cells during the next time step (assuming traffic is not congested), in which the next set of new vehicles are added to the array. Thus, it is not possible with this model to determine where the maximum occurs when `v_max` > 1.

5.4 Distribution of a roadblock

The results obtained in Sec. 4.4 point to the conclusion that a roadblock of length smaller than the value of `v_max` does not affect the free traffic flow. It is possible that this result is a consequence of the fact that vehicles are able to change lanes with no penalty to their velocity and without disturbing surrounding traffic. A different lane-changing model might give very different results during roadblock simulations, since the modeling of lane-changing is such an integral part of this traffic scenario.

The results also suggest that when the roadblock length passes a certain threshold (roughly 14 cells here), further increase in the length does not affect the vehicle density significantly. It is possible that this is because the density began to approach saturation, due to the finite amount of cells in the array.

5.5 A line of cars behind a traffic light

Figures 11 and 13 corresponding to the Newell model passing times show mostly linear relationships between the model parameters and the passing time. This seems in line with the linear nature of the Newell v_{opt} model. The results for the Gipps model, as seen in Figures 10 and 12, show relationships that differ from the Newell model. For the Gipps model, it seems that the relationships approach the linear case as the reaction time or the number of vehicles increases, though they start out with exponential characteristics.

The data points show that in the Gipps model, the vehicles pass the traffic light much faster than in the Newell model. With the reaction time $T = 1$ s, the Gipps model is almost 7 times faster than the Newell model. With 5 vehicles, the Gipps model is just over 6 times faster. As the two parameters are increased, however, the percental difference begins to diminish. With $T = 8$ s, the Gipps model is almost 2 times faster. With 40 vehicles, the Gipps model is just over 3 times faster.

These results are useful in determining how long traffic lights at an intersection should be green. In a realistic scenario, we have two roads at an intersection with a traffic light, where there may be a difference in vehicle flow between the roads. In that scenario, it is desirable to determine how long each green time period should be in order to maximize vehicle flow. This particular scenario is difficult to simulate with the implementation of car-following models that was used. What the results tell us, though, is that the Gipps model indicates that a line with 10 cars passes the light more than twice as fast as a line with 20 cars. Thus, it seems that it would be more beneficial to let the traffic lights switch between green and red often, rather than keeping the light green for a long period, followed by a long period of red. In this way, cars may pass before the line grows too large and the passing times increase too much. Of course, it is not possible to draw this conclusion solely based on these simulations, the conclusion is merely an indication.

References

[1] Dirk Helbing, 2001, *Traffic and Related Self-Driven Many-Particle Systems.*

[2] Martin Treiber and Arne Kesting, 2013, *Traffic Flow Dynamics: Data, Models and Simulation*, Berlin, Heidelberg: Springer, E-ISBN 978-3-642-32460-4.

[3] Stephen Wolfram, 1983, *Statistical mechanics of cellular automata*, Reviews of Modern Physics, Vol. 55 (3), 601-644.