Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Modelling the phenomenon of congestion at Gotthard

A case study

Eric Hayoz, Janick Zwyssig

Zurich
December 2014

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Eric Hayoz	Janick Zwyssig
Eric Hayoz	

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

Modelling the phenomenon of congestion at Gotthard - A case study

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
Hayoz	Eric
Zwyssig	Janick

Supervising lecturer

Last name	First name
Kuhn	Tobias
Woolley	Olivia

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Eric Hayoz

Zurich, 12. December 2014 — Place and date

Eric Hayoz   Janick Zwyssig — Signature

Contents

1 Abstract

Abstract of Eric Hayoz & Janick Zwyssig, Modelling the phenomenon of congestion at Gotthard – A case study, 2014:
This work describes the modelling of congestion on the stage between Erstfeld and the Gotthard tunnel in Göschenen, in the canton of Uri. In a first step a simulation based on the Nagel-Schreckenberg model was created. The model was supplemented by a second lane, a red-light and the ability to change the lane. A mapping from real data to the model allows to compare values. The main goal is to train the model that one can compare the measured congestion to a real traffic jam. For that the authors have implemented a congestion measurement and have processed datasets, provided by ASTRA (Amt für Strassen). A parameter setting that fits the reference data best was found.
Finally the trained model was feed by two other, unused datasets to compare the results. Unfortunately, due to abnormalities of the datasets and some restrictions of the model there are big deviations to reference data.

2 Individual contributions

We devided the code work as follows:
Eric Hayoz:

- Implementation NaSch-Modell for 2 cars
- Handling Dataset with inFlow and Speed
- Correlation Reality - Model
- Creating optiFinder

Janick Zwyssig:

- Implementation NaSch-Modell for n cars
- Implementation laneChange & redLight
- curve fitting for results
- congestion measurement

In the end both authors contributed to every section. The article was written together.

3 Introduction

Nowadays, traffic jams belong to every day life. Because Switzerland is an important transit way to Southern Europe for traffic, the Gotthard tunnel is an unavoidable bottleneck. The tunnel stretches over 15 Kilometers from Göschenen, Kanton Uri to Airolo in the canton of Ticino. Over and over again it comes to long traffic jams, at peak times up to 20 Kilometers. Responsible for that are on the one hand holidays in Switzerland or neighbor countries and on the other hand the lane reduction from two to one lane. Switzerland has been discussing this issue for many years and the construction of a second tunnel was proposed. The purpose of the second tunnel should be built for security reasons and not in order to increase the traffic flow. This is due to an election 20 years ago, called the Alpenschutzinitiative. In the near future there will be a restauration of the tunnel. That's why the discussion about a second tunnel will commence once again.
The authors have the goal to investigate the Gotthard tunnel on the north side in Göschenen. That's why a model has been created. By means of the number of cars passing through Erstfeld (a village 19 Kilometers from the tunnel), the length of congestion can be measured. The Nagel-Schreckenberg model will be employed for the simulation.

3.1 Motivation

As one of the authors is born in the canton of Uri he knows about the traffic issues at the Gotthard tunnel. This explains the personal motivation to investigate the phenomenon of traffic congestions. The other author is often on the way by car, he has already wasted a lot of time in traffic congestions and thus has also have burning interest in efficient solutions to handle them.

4 Description of the Model

4.1 The Nagel-Schreckenberg model

This model was created in the 90s by the German physicists Kai Nagel and Michael Schreckenberg to simulate traffic flow on highways. It explained the so called *Phantom-Congestion* which occurs when drivers linger by not accelerating even if they have the opportunity to do so.
The model is a cellular automata based on four very simple rules, which are applied in sequence. One complete loop is called an iteration.

1. Acceleration: Each vehicle, that has not yet reached the maximum speed (which is 5), increases its speed by one unit.
2. Deceleration: If the size of the gap (in cells) between two vehicles is less than the speed of the following car (in speed units), this cars speed is reduced to a value equivalent to the size of the gap.
3. Lingering: Every vehicle reduces its speed by 1 with a probability p ($0 \leq p \leq 1$).
4. Moving forward: Every vehicle moves forward the number of cells equal to its speed.

4.2 Example of an iteration

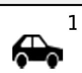
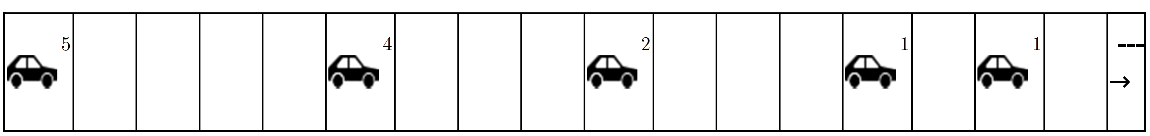
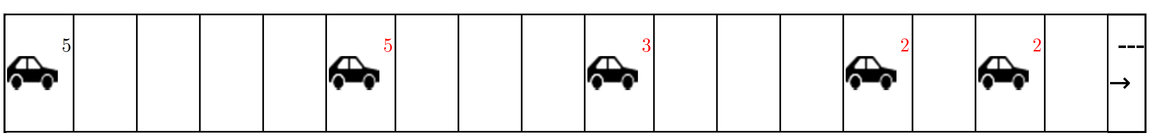
Symbol	Meaning
	A vehicle with a speed of 1 (27 km/h)
	A vehicle that has accelerated or decelerated to a speed of 3 (81 km/h)
	A vehicle has moved with a speed of 2 (going 2 cells forward)

Figure 1: Nagel-Schreckenberg model – Symbols

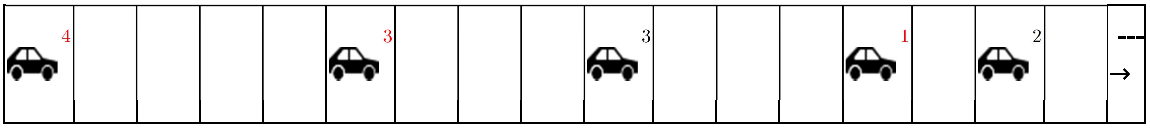
Random startup setting for time t



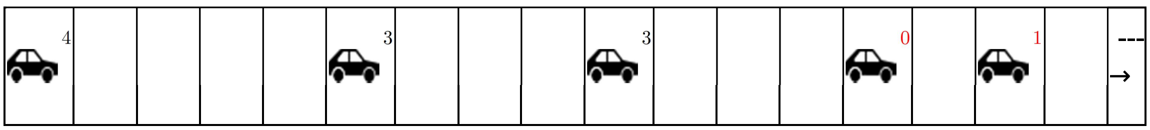
Step (1) – Acceleration, $v_{max} = 5$



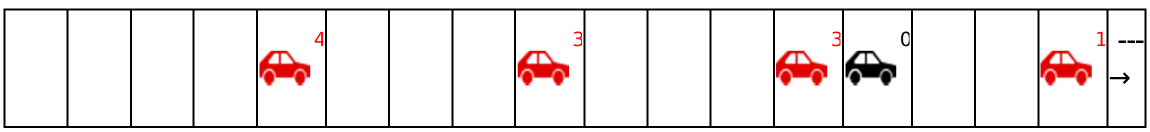
Step (2) – Deceleration



Step (3) – Lingering ($\rho = 1/3$, affecting the two leading cars)



Step (4) – Moving (Setting for the time $t + 1$)



4.3 Situation

The distance between Erstfeld and the Gotthard tunnel is relevant for the model. The highway to the tunnel is two-laned and will be reduced to one lane in front of the tunnel. As on the figure 2 shown, the authors resigned to depicted any exits. The reason for this is first to reduce complexity of the model and second to neglect irrelevant factors. In case of congestion there is a red light that regulates the traffic flow, which is also taken into account in the model.

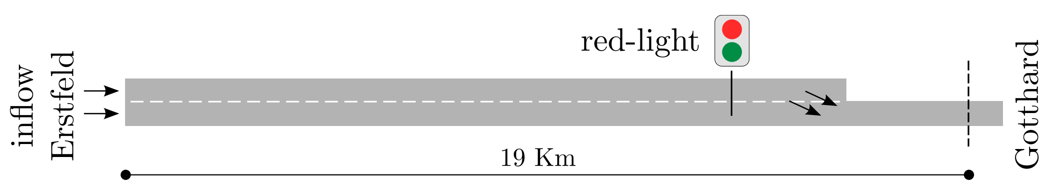


Figure 2: Sketch of model

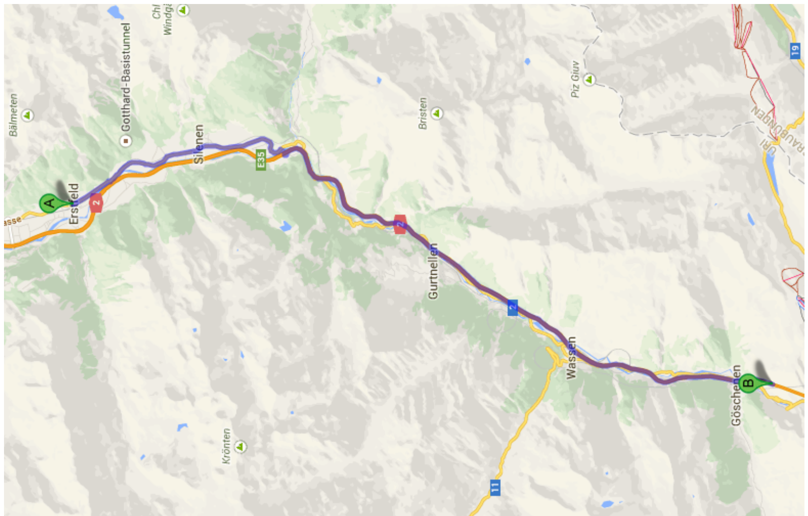


Figure 3: map of Uri, A: Erstfeld, B: Gotthard tunnel at Göschenen

5 Implementation

5.1 Correlation between reality and model

Item	Reality	Model
Time	1s	1 iteration
Maximum speed	120 km/h (we defined vmax as 100 km/h to get even values)	5 (only 6 levels)
Length of an average car (Skoda Octavia)	4.5 m	1 cell
Distance Erstfeld - Göschenen	19’000 m	4222 cells (19’000 m / 4.5 m, rounded)

Figure 4: Units

Variable name	Meaning	Values
moveProb	Probability for a car to move forward	0. . . 1
moveCorr	Modify moveProb for a given time (in order to improve congestion length prediction)	-1. . . 1
laneChange	Probability for a car to change its lane	0. . . 1
-	Value of laneChange as a function of distance to changeCell (for locations where cars have to change from left to right)	Constants in equation
redLight	Illuminate a dropcounter redlight in front of tunnel (it turns on if congestion length > 2 km and redLight is set to 1)	0, 1
dropCounter	Length of a period when redlight is ON	0. . . Inf
-	Maximum speed just in front of tunnel	0. . . 5

Figure 5: Variables

5.2 Inflow

The inflow is given in #cars / 180 s. For every iteration in our model, the cars are spawned in Erstfeld (starting position) on the two lanes with a total probability p = #cars/180. As there are normally slightly more vehicles on the right lane, we multiply p for the right lane with a factor of 1.1 and the p for the left lane with a factor of 0.9.

5.3 Average speed

First we convert the average speed from km/h (0. . . 120) to “Nagel-Schreckenberg-Speed” (0. . . 5). If the speed is more than 5, we round down to 5. Example:
Average Speed is 94 km/h. Converting into Nagel-Schreckenberg-Speed results in 4.7. Our model would in this case spawn 70% of the cars with a speed of 5, and 30% with a speed of 4.

5.4 Lane Change and red-light

As cars change lane often in reality, it should be also a part of the model. Furthermore, it is necessary to change the lane due to lane reduction in front of the tunnel. The probability to change the lane from left to the right is higher than the other way around. That yields more cars on the right lane. In front of the tunnel the probability for a lane change increases, due to lane reduction.
Red-light is activated if the congestion length exceed the length of one Kilometre. It’s acting approximately 40 meters in front of the lane reduction. Every second one car can pass through the tunnel.

5.5 Congestion Length

The length of a congestion is given in Kilometres. First we calculate the length of the congestion in our simulation and then multiply the length (unit: cells) by the length of a cell (4.5 m) to enable a comparison between the virtual (calculated) congestion with the real (measured) congestion.
Measuring the congestion length in our model: To obtain a stable output, we divide the highway into blocks with a length of 50 cells. If a block fulfils the two following criteria, it is declared a congestion block: high density and low average speed. We count the congestion blocks starting in Göschenen heading northbound to Erstfeld, and we only accept one uncongested block between any two blocks, otherwise we consider the congestion as terminated at the point where two or more blocks do not fulfil the congestion criteria anymore.

5.6 Processing the datasets delivered by ASTRA

The attached extract shows among other things that we have information about the inflow values and average speeds in 3 minutes intervals. We also have the length of the congestion at specific times (random times and course measurements). These are the three values that we used.

Datum	Fahrzeuge pro 3Min	Durchschnittsgeschwindigkeit	Fahrzeuge pro 3Min Vergleichstage Fr 25.7.2014 - Sa 26.7.2014	Durchschnittsgeschwindigkeit Vergleichstage Fr 25.7.2014 - Sa 26.7.2014	Fahrzeuge pro 3Min Vergleichstage Fr 1.8.2014 - Sa 2.8.2014	Durchschnittsgeschwindigkeit Vergleichstage Fr 1.8.2014 - Sa 2.8.2014</title>
2014-07-18 00:00-00:03	30	114	22	110	33	110
2014-07-18 00:03-00:06	22	116	15	98	37	107
2014-07-18 00:06-00:09	22	115	25	116	42	115

Figure 6: Inflow values with average speeds

Datum	Uhrzeit	zwischen AS	Name	und AS	Name	Stau-km	Warte-zeit	Verkehrs-information	Empfehlung
02.08.2014	05:10	39	Wassen	40	Göschenen			stockender Verkehr	keine
02.08.2014	05:30	39	Wassen	40	Göschenen	2	20	Stau, Verkehrsüberlastung	keine
02.08.2014	06:11	39	Wassen	40	Göschenen	3	30	Stau, Verkehrsüberlastung.Einfahrt Gö gesperrt.	keine

Figure 7: Congestion data

6 Simulation Results and Discussion

The reference data for congestion measurement is provided by ASTRA (Bundesamt für Strassen). There are four datasets:

- Dataset 1: measurement of 2014 July, 18. - 19.
- Dataset 2: measurement of 2014 August, first
- Dataset 3: measurement of 2014 July, 25. - 26.
- Dataset 4: measurement of 2014 August, second

In a first step the model will be trained by dataset 1 and 2. The authors wrote a program which calculates specified values of a specified variable (moveProb and moveCorr) for a given number of times. With other words, it finds the optimal parameter values that minimize the error of measured congestion. In a second step, the datasets 3 and 4 are used to evaluate the previous trained model.

6.1 Preparing dataset – curve fitting

The timestamps for the measurement are irregular. In order to compare the congestion from the model to the reference data, the breakpoints will be interpolated. The resulting curve can be used to generate regular timestamps for measurement. The authors have opted for a linear interpolation. That means that every point will be connected by a straight line with its neighbour, see on the figures below. A linear interpolation is reasonable because the congestion length between two breakpoints cannot change very quickly.

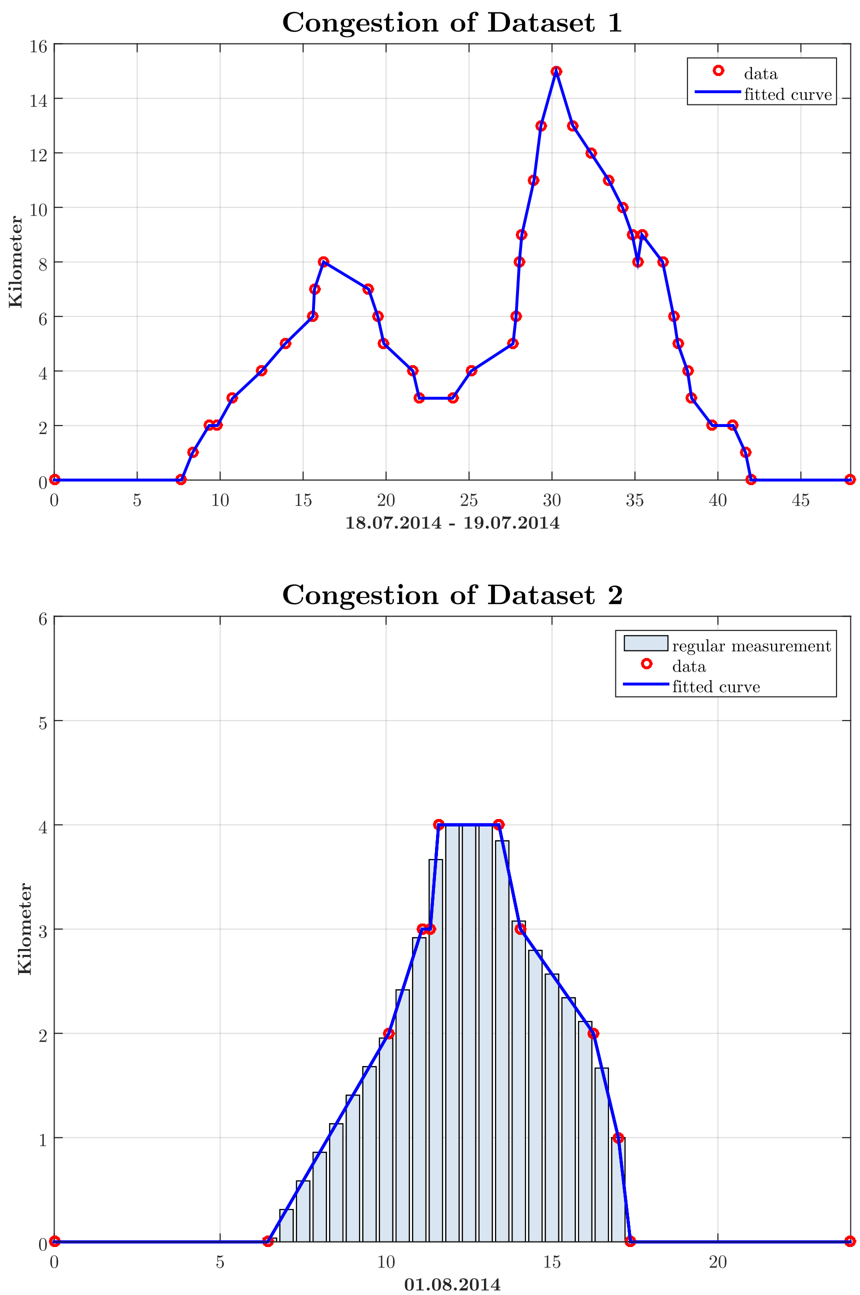


Figure 9: Dataset 2, added regular measurements for comparison

6.2 Train model

There are two parameters to fit the model:

- moveProb: describes how accurate cars moving forward
- moveCorr: increases moveProb in case of congestion

The program optiFinder was written to find the best setting of these two parameters. Important here by are the quantities *prediction* and *precision*. The latter describes the relative ratio between the area outside the intersection and the area of the reference data itself. One can imagine two portions A and B and its intersection $A \cap B$. The precision would be precision $= 1 - (A \cup B - A \cap B)/A$, where A is the area from the reference data.

Prediction is the ratio between all x-values that are inscribed by the intersection of the two curves and the reference curve. The big difference to precision is the way of counting. It would be very beneficial for the prediction value, if there is a lot of congestion. The outcome of this would be a poor precision value. That's why precision and prediction are used to find the optimal parameter. Because precision is more important, the weight for it is three times greater than for prediction.

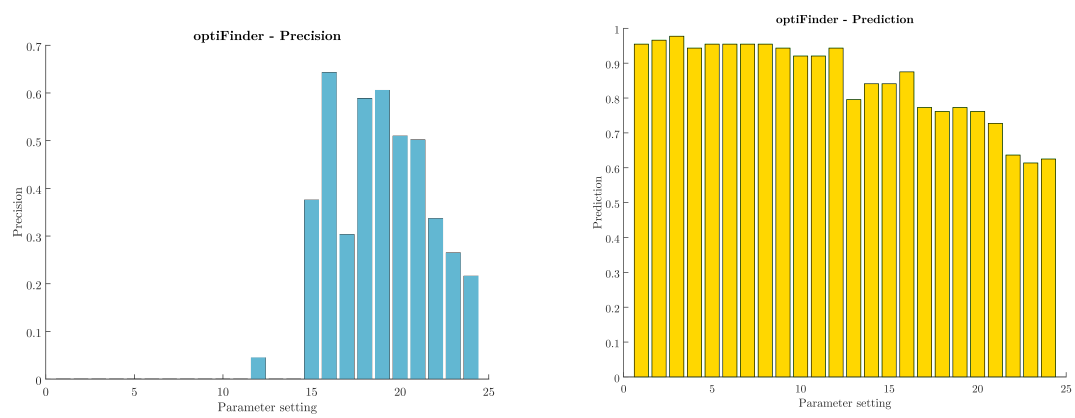


Figure 10: left: precision, right prediction for Dataset 2

The weighting of prediction and precision lead to the values for moveProb and moveCorr.

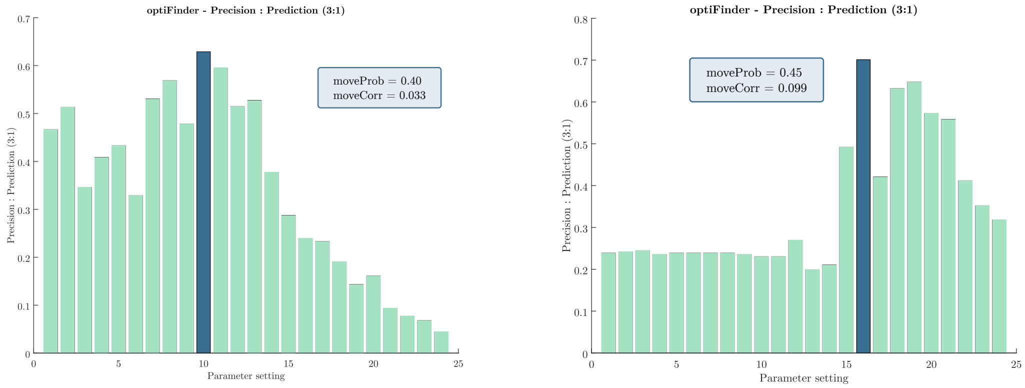


Figure 11: left: Dataset 1, right: Dataset 2, values from prediction (1x) and precision (3x)

The mean of the two found parameters are:
$moveProb = 0.425$ and $moveCorr = 0.066$.

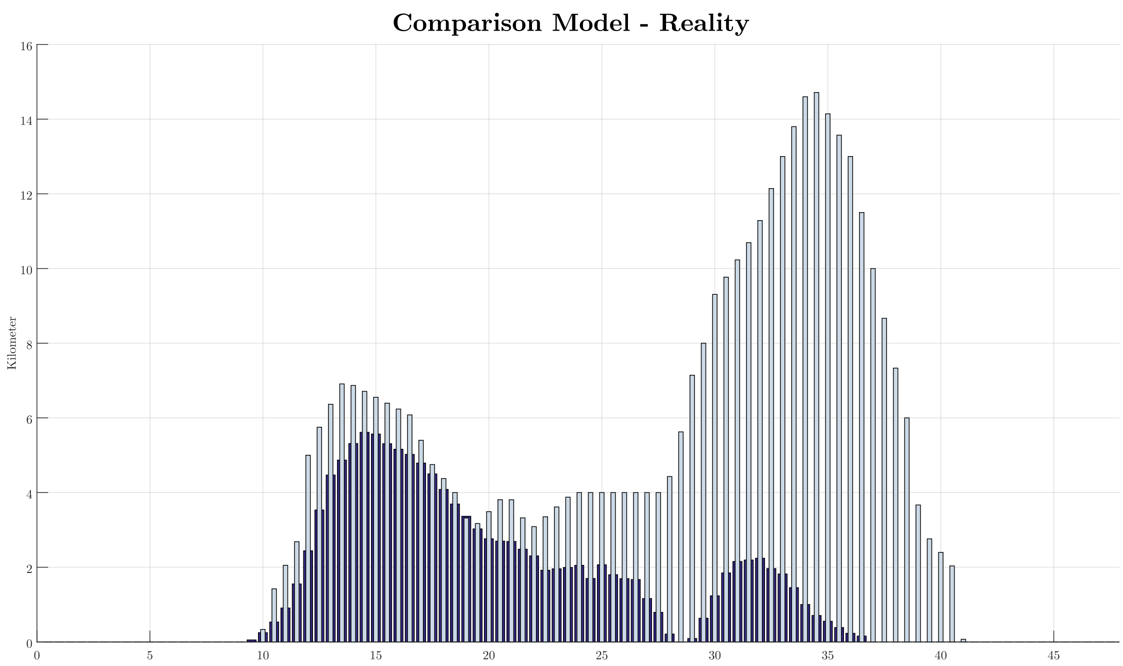


Figure 12: Dataset 3 – precision = 0.31, prediction = 0.84

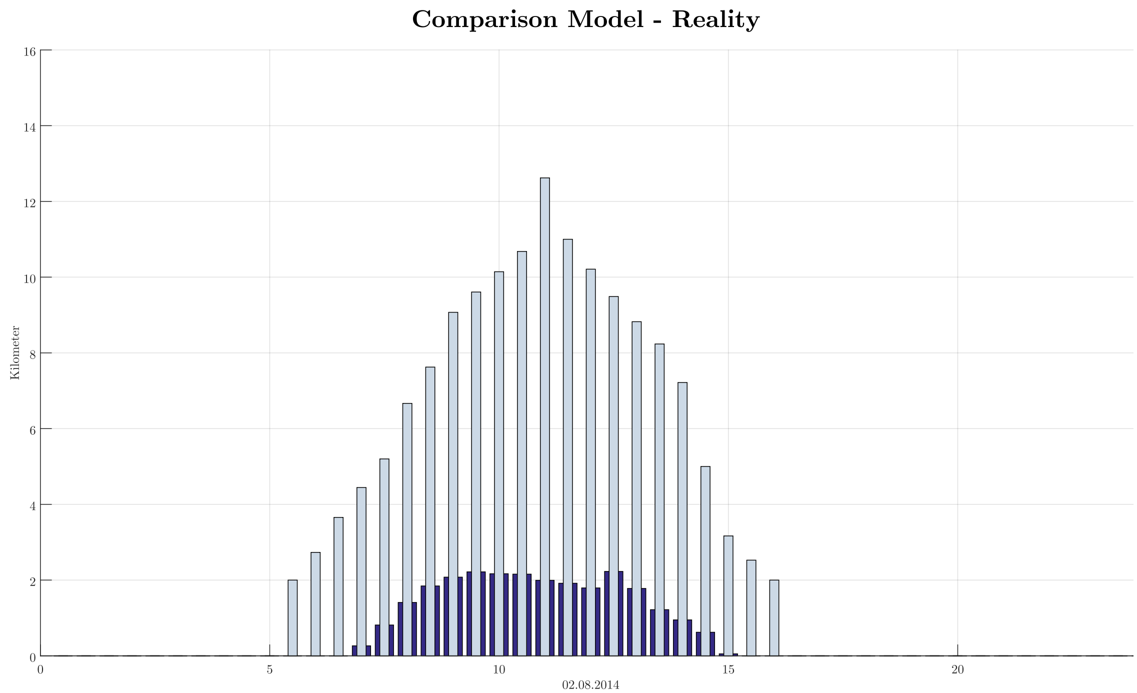


Figure 13: Dataset 4 – precision = 0.17, prediction = 0.77

7 Summary and Outlook

The authors created a model based on the Nagel-Schreckenberg model. It was extended to fit the situation at the Gotthard-Strassentunnel Nordportal between Erstfeld and Göschenenen. Its purpose is to predict traffic congestions in front of the tunnel.
The model was firstly trained with two datasets (24h and 48h), which were both provided by ASTRA.
The authors soon realised that there were some uncasualities in the congestion plot diagrams, which lead back to measuring mistakes or inaccuracy of the dataset. They had to adjust these abnormalities in order to obtain appropriate output.
At the very end, the finished model had to prove its accuracy by predicting traffic congestions with two other datasets (24h and 48h, also provided by ASTRA), which have not been used before.

7.1 Outlook

Both of the authors invested much time and effort in this project, and they really liked working on it. It is indisputable that some misleading measurements from the datasets distorted the results.
Without a doubt systems of this nature will play an important role in the future.

8 Appendix

References

[1] Andreas Schadschneider. *Traffic flow modelling*. `http://www.thp.uni-koeln.de/~as/Mypage/traffic.html`. April 2000

[2] Torsten Held, Stefan Bittihn. *Cellular automata for traffic simulation – Nagel-Schreckenberg model*. March 2011

[3] K. Nagel and M. Schreckenberg. *A cellular automaton model for freeway traffic*. 1992

List of Figures

Code

Listing 1: interpolate

```
function [Y_congestion, data] = interpolate(dataset, interval, showPlot)
%% parameters
% INPUT:
%   dataset:        choose one of 4 datasets, {1,2,3,4}
%   interval:       interval of measured congestion in minutes
%   showPlot:       show interpolated curve, {0=false, 1 = true}
% OUTPUT:
%   Y_congestion:   length of congestion, #measures depend on interval
%   data:           vector, contains time data

%% Prepare data (ASTRA)

switch dataset
    case 1
        time = {'18.07.2014 00:00' '18.07.2014 07:40' '18.07.2014 08:20' '
            18.07.2014 09:20' '18.07.2014 09:50' '18.07.2014 10:45' '18.07.2014
            12:30' '18.07.2014 13:55' '18.07.2014 15:36' '18.07.2014 15:41' '
            18.07.2014 16:14' '18.07.2014 18:57' '18.07.2014 19:30' '18.07.2014
            19:52' '18.07.2014 21:37' '18.07.2014 22:00' '19.07.2014 00:00' '
            19.07.2014 01:08' '19.07.2014 03:39' '19.07.2014 03:50' '19.07.2014
            04:02' '19.07.2014 04:11' '19.07.2014 04:55' '19.07.2014 05:20' '
            19.07.2014 06:15' '19.07.2014 07:15' '19.07.2014 08:20' '19.07.2014
            09:25' '19.07.2014 10:15' '19.07.2014 10:50' '19.07.2014 11:10' '
            19.07.2014 11:25' '19.07.2014 12:40' '19.07.2014 13:20' '19.07.2014
            13:35' '19.07.2014 14:10' '19.07.2014 14:25' '19.07.2014 15:40' '
            19.07.2014 16:55' '19.07.2014 17:40' '19.07.2014 18:00' '19.07.2014
            23:59'};
        congestion = [0 0 1 2 2 3 4 5 6 7 8 7 6 5 4 3 3 4 5 6 8 9 11 13 15 13 12
             11 10 9 8 9 8 6 5 4 3 2 2 1 0 0];
    case 2
        time = {'01.08.2014 00:00' '01.08.2014 06:26' '01.08.2014 10:05' '
            01.08.2014 11:05' '01.08.2014 11:20' '01.08.2014 11:35' '01.08.2014
            13:24' '01.08.2014 14:03' '01.08.2014 16:15' '01.08.2014 17:00' '
            01.08.2014 17:22' '01.08.2014 23:59'};
        congestion = [ 0 0 2 3 3 4 4 3 2 1 0 0];
    case 3
        time = {'25.07.2014 00:00' '25.07.2014 09:55' '25.07.2014 10:10' '
            25.07.2014 11:45' '25.07.2014 11:55' '25.07.2014 12:00' '25.07.2014
            12:40' '25.07.2014 13:35' '25.07.2014 16:45' '25.07.2014 17:10' '
            25.07.2014 18:30' '25.07.2014 19:14' '25.07.2014 20:48' '25.07.2014
            21:50' '25.07.2014 23:44' '26.07.2014 00:00' '26.07.2014 01:57' '
            26.07.2014 02:38' '26.07.2014 03:45' '26.07.2014 04:20' '26.07.2014
            04:36' '26.07.2014 04:55' '26.07.2014 05:30' '26.07.2014 05:40' '
            26.07.2014 07:50' '26.07.2014 09:00' '26.07.2014 10:15' '26.07.2014
            12:00' '26.07.2014 13:00' '26.07.2014 13:45' '26.07.2014 14:30' '
            26.07.2014 14:55' '26.07.2014 15:10' '26.07.2014 16:33' '26.07.2014
            16:47' '26.07.2014 17:01' '26.07.2014 23:59'};
        congestion = [0 0 1 3 4 5 6 7 6 5 4 3 4 3 4 4 4 4 4 5 6 7 8 9 11 13 15
            13 10 8 6 4 3 2 1 0 0];
    case 4
        time = {'02.08.2014 00:00' '02.08.2014 05:10' '02.08.2014 05:30' '
```

```
            02.08.2014 06:11' '02.08.2014 06:40' '02.08.2014 07:25' '02.08.2014
            07:50' '02.08.2014 08:05' '02.08.2014 08:45' '02.08.2014 08:56' '
            02.08.2014 10:48' '02.08.2014 10:53' '02.08.2014 11:30' '02.08.2014
            12:08' '02.08.2014 12:51' '02.08.2014 13:42' '02.08.2014 14:05' '
            02.08.2014 14:15' '02.08.2014 14:30' '02.08.2014 14:45' '02.08.2014
            15:03' '02.08.2014 16:00' '02.08.2014 16:17' '02.08.2014 23:59'};
        congestion = [ 0 0 2 3 4 5 6 7 8 9 11 13 11 10 9 8 7 6 5 4 3 2 0 0];
    otherwise
        disp (['dataset ' num2str(dataset) ' does not exist. Available datasets:
             training(1, 2); evaluation(3,4)'])
        return
end

% generate date vector
data = datevec(time, 'dd.mm.yyyy HH:MM');

% compute #seconds since 00:00 of first day
seconds = zeros(1, length(time));
first_date = datenum(data(1,1:3));
for i = 1:length(time)
    curr_date = datenum(data(i,1:3));
    if curr_date == first_date
       seconds(1,i) = data(i,4)*3600 + data(i,5)*60;
       xMax = 24;
    else
       seconds(1,i) = data(i,4)*3600 + data(i,5)*60 + 24*3600;
       xMax = 48;
    end
end

% data for curve fitting
X = seconds / 3600;
Y = congestion;

%% Curve Fitting
[xData, yData] = prepareCurveData(X, Y);

% Set up fittype and options.
ft = 'linearinterp';
opts = fitoptions(ft);
opts.Normalize = 'on';

% Fit model to data.
[fitresult] = fit(xData, yData, ft, opts);

if showPlot
   % Plot fit with data.
   figure('Name', ['Dataset ' num2str(dataset)]);
   p = plot(fitresult, '-b', xData, yData, 'or');
   set(p, 'LineWidth', 2)

   % Label axes
   ylim([0 16]);
   if mod(dataset,2)
      xlabel([datestr(data(1,:),'dd.mm.yyyy') ' - ' datestr(data(end,:),'dd.mm.
          yyyy')],'fontweight','bold','fontsize',11);
   else
      xlabel(datestr(data(1,:),'dd.mm.yyyy'),'fontweight','bold','fontsize',11);
```

```
        if dataset == 2
            ylim([0 6]);
        end
    end
    xlim([0 xMax]);
    ylabel('Kilometer', 'fontweight','bold','fontsize',11);
    t = title(['Congestion of Dataset ' num2str(dataset)]);
    set(t,'fontweight','bold','fontsize', 18);
    grid on
end
%% Generate congestion data for error-evaluation
Y_congestion = fitresult(interval/60:interval/60:xMax);
end
```

Listing 2: optiFinder

```
function [moveProb, moveCorr] = optiFinder(dataset)
% Optimum Finder for:
%   - moveProb
%   - moveCorr

%% moveCorr
mc_start = 0;
mc_stop = .099;
mc_step = .033;

%% moveProb
mp_start = .3;
mp_stop = .55;
mp_step = .05;

isAnimated = 0;
rounds = 4;
numberMC = round((mc_stop-mc_start)/mc_step+1);
numberMP = round((mp_stop-mp_start)/mp_step+1);
evaluation = zeros(2,numberMC * numberMP);
precision_tot = 0;
prediction_tot = 0;
v = 0;

set(0,'DefaultFigureVisible','off') % suppress bar graph output
for moveProb = mp_start:mp_step:mp_stop
 for moveCorr = mc_start:mc_step:mc_stop
     v = v+1;
   for j = 1:rounds
       [precision,prediction]  = NaSch_Datasets_v1(dataset, moveProb, isAnimated
           , moveCorr);
       precision_tot = precision_tot + precision;
       prediction_tot = prediction_tot + prediction;
   end
   evaluation(1,v) = precision_tot/rounds;
   evaluation(2,v) = prediction_tot/rounds;
   precision_tot = 0;
   prediction_tot = 0;
 end
end
% diagram of different errors
set(0,'DefaultFigureVisible','on') % do not suppress following bar graph output
figure()
hold on;
title('optiFinder - Precision')
x = 1:length(evaluation);
y = evaluation(1,:);
xlabel('Parameter setting');
ylabel('Precision');
bar(x,y, 'EdgeColor','g', 'FaceColor','g')

figure()
hold on;
title('optiFinder - Prediction')
x = 1:length(evaluation);
y = evaluation(2,:);
```

```
xlabel('Parameter setting');
ylabel('Prediction');
bar(x,y, 'EdgeColor','g', 'FaceColor','g')

figure()
hold on;
title('optiFinder - Precision : Prediction (3:1)')
x = 1:length(evaluation);
y = (3*evaluation(1,:)+evaluation(2,:))/4;
xlabel('Parameter setting');
ylabel('Precision : Prediction (3:1)');
bar(x,y, 'EdgeColor','g', 'FaceColor','g')

[Max, Index] = max(y);
T = zeros(numberMC, numberMP);
T(Index) = 1;
[value, location] = max(T(:));
[R,C] = ind2sub(size(T),location);
moveProb = mp_start + (C-1) * mp_step;
moveCorr = mc_start + (R-1) * mc_step;
end
```

Listing 3: model.m

```
function [precision, prediction] = NaSch_Datasets_v1(dataset, moveProb, isAnimated, moveCorr)

% values
% NaSch_Datasets_v1(1,.525,0,0,0,.055): perfectly symmetric
% NaSch_Datasets_v1(2,.525,0,0,0,.05):  less error

%% parameters
% INPUT:
%   dataset: choose one of 4 datasets, {1,2,3,4} (don't use 3,4:evaluation)
%   moveProb: the probability for a car to move forwards, 0..1
%   isAnimated: start program with animation, boolean 1=true 0=false
% OUTPUT:
%   error_tot:  total
%
% EXAMPLE:
%   NaSch_1C_Stats_v1(2, .5, 0 , 1, 0)

% parameters for comparison model - reality
lC = 4.5;                 % length of each cell (average length of cars => Skoda Octavia)
lR = 19000;               % length in Reality (Erstfeld - Goeschenen, 13min)
N = round(lR / lC);       % 4222 cells
measureInterval = 30;     % measure every #min
[I, S] = Datasets(dataset);
nIter = length(I)*180;    % number of iterations; datasets: #cars/180s => we want 1 iteration / s
q = 0;                    % running variable for reading outSet
redLight_act = 1;         % activate redLight

% set parameter values
conv = 1000/lC;     % "convert", #cells that matches 1km
cC = N-22;          % "change cell", where cars have to change the lane
vmax = 5;           % maximal velocity of the cars (vmax = 5 = 100 km/h)
L = 11;             % length of lane where cars can change (in front of cC)
vmax_L = 3;         % maximal velocity in L
a = 0.2;            % min probabilty that car changes lane at cC - L
b = 0.6;            % max probabilty that car changes lane at cC
laneChange = .1;    % the probability for a car to change the lane, 0..1
dropCounter = 2;    % dropCounter: #seconds a car can pass the redlight, redlight is active
                    % after 2km congestion. Use '1' to turn off redlight. Do NOT
                    % use odd numbers.
redLight = 0;       % automatic redLight, do NOT turn on
inflowCounter = 0; % count cars

% use quadratic increments for the probabilty between a and b (p = k*x^2+d)
k = (a - b)/(L*L-2*cC*L);            % for x=cC-L is p = a
d = b - cC*cC*(a - b)/(L*L-2*cC*L);  % for x=cC   is p = b

% set statistical variables
vSum = 0;        % sum of speeds
nCars = 0;       % #cars on road

% define variables in a block (2 x bL)
bL = 50;         % block length:   length of a block // 225m
```

```
bD = 0;          % block density:  density of cars in a block, 0..1
bV = 0;          % block velocity: average velocity in a block, 0..vmax
bC = 0;          % block counter:  counts number of blocks (congestion), 0..(N %
     bL)
bE = 0;          % empty counter:  counts number of blocks (no congestion),
    {0,1,2}

% congestion length in each round
if mod(dataset,2)
    divider = 48*60 / measureInterval;
else
    divider = 24*60 / measureInterval;
end
congLength = zeros(1,nIter/divider);
congPlot = zeros(1,divider);
currentCongestion = 0;
first = 1;                          % congestion optimization
opt_act = 0;
congStart = 0;

% define road (-1 = no car; 0..vmax = car, value represents velocity)
X = -ones(2,N);

% take average inflow (every 2 hours)
if mod(dataset,2)
    inflow = zeros(1,24);
    for i = 0:23
        inflow(1,i+1) = sum(I(1,1+i*40:40+i*40)) / 40;
    end
else
    inflow = zeros(1,12);
    for i = 0:11
        inflow(1,i+1) = sum(I(1,1+i*40:40+i*40)) / 40;
    end
end
p=1;

%% main loop, iterating the time variable, t
for t = 1:nIter
    %% NaSch and laneChange
    % cars change lane with given probability laneChange
    for i = 1:cC-L
        % left to right --> probabilty laneChange
        if X(1,i) ~= -1  &&  X(2,i) == -1  &&  rand < laneChange
            X(2,i) = X(1,i);
            X(1,i) = -1;
        end
        % right to left --> probabilty 0.95*laneChange
        if X(2,i) ~= -1  &&  X(1,i) == -1  &&  rand < 0.95*laneChange
            X(1,i) = X(2,i);
            X(2,i) = -1;
        end
    end

    % acceleration (NaSch -- RULE 1) ================
    for i = 1:2*N
        if X(i) ~= -1  &&  X(i) < vmax
            X(i) = X(i) + 1;
```

```
        end
    end

    % cars have to change lane due to lane reduction
    for i = cC-L:cC

        % reduce velocity of lanes in front of lane reduction
        if X(1,i) > vmax_L
            X(1,i) = vmax_L;
        elseif X(2,i) > vmax_L
            X(2,i) = vmax_L;
        end

        % lane change on road
        if X(1,i) ~= -1 % change lane if possible
            if i == cC  &&  X(2,i) == -1
                X(2,i) = X(1,i) + 2;      % accelerate after lane change
                X(1,i) = -1;
            elseif X(2,i) == -1 && rand < (k*i*i + d)
                X(2,i) = X(1,i) + 2; % +2 accelerate
                X(1,i) = -1;
            elseif X(1,i)+i > cC % avoid overrunning changeCell
                X(1,i) = cC-i;
            end
        end
    end

    % red light
    if X(1,cC-10) ~= -1 && redLight
        X(1,cC-10) = 0;
        if mod(t+dropCounter/2,dropCounter) == 0
            X(1,cC-10) = 2;
        end
    end

    if X(2,cC-10) ~= -1 && redLight
        X(2,cC-10) = 0;
        if mod(t,dropCounter) == 0
            X(2,cC-10) = 2;
        end
    end

    % slowing down (NaSch -- RULE 2) ===============
    for row = 1:2
        for i = 1:N
            if X(row,i) ~= -1
                for j = 1:X(row,i)
                    if i+j <= N
                        if X(row,i+j) ~= -1
                            X(row,i) = j-1;
                            break
                        end
                    end
                end
            end
        end
    end

```

```
    % randomization (NaSch -- RULE 3) ===============
    for i = 1:2*N
        if X(i) > 0
            X(i) = X(i) - (rand > moveProb);
        end
    end

    % car motion (NaSch -- RULE 4) ==================

    %% inflow
    % update positions X(1..N)
    Xold = X;
    for row = 1:2
        for i = 1:N
            if Xold(row,i) > 0 && i + Xold(row,i) <= N
                X(row,i+X(row,i)) = X(row,i);
                X(row,i) = -1;
            elseif Xold(row,i) > 0 && i + Xold(row,i) > N
                X(row,i) = -1;
            end
        end
    end

    if t > q*180 % datasets: #cars / 180s => every 180s we take new inflow value from dataset
        q = q + 1;

        % average inflow
        if mod(q,40) == 0
           p = p + 1;
        end

        if mod(dataset,2)
            if p > 24
               p = 24;
            end
        else
            if p > 12
               p = 12;
            end
        end
        rateI = inflow(1,p)/(2*180);

        rateS_m = S(1,q); % mean speed
        rateS_m = rateS_m/(3.6*5.55555); % convert km/h into NaSch-units
        if rateS_m > 5
            rateS_m = 5;
        elseif rateS_m < 2
            rateS_m = 2;
        end
    end

    rateS = ceil(rateS_m) - (rand < (ceil(rateS_m)-rateS_m));

    % update position X(1,1) left lane (inflow left)
    % calculate inflow rate per second, divide by 2 because
    % the 2 rows, multiply by .95 because left lane
    rate = rateI*.95;
```

```
    if rand < rate && X(1,1) == -1
        X(1,1) = rateS;  % all cars enter with speed of rateS
        inflowCounter = inflowCounter + 1;
    end

    % update position X(2,1) right lane (inflow right)
    rate = rateI*1.05;

    if rand < rate && X(2,1) == -1
        X(2,1) = rateS;  % all cars enter with speed of rateS
        inflowCounter = inflowCounter + 1;
    end

    %% statistics
    % average speed (only with animation)
    if isAnimated
        for row = 1:2
            for i = 1:cC
                if X(row,i) ~= -1
                    vSum = vSum + X(row,i);
                    nCars = nCars + 1;
                end
            end
        end

        vAverage = vSum / nCars;
        % reset vSum for next round
        vSum = 0;
        nCars = 0;
    end

    % congestion length
    for i = cC:-bL:1
        % compute density of cars and average velocity in a block
        for j = 2*i:-1:2*(i-bL) + 1
            if X(j) ~= -1
                bV = bV + X(j);
                bD = bD + 1;
            end
        end
        if bD == 0
            bD = 1;
        end
        bV = bV / bD;
        bD = bD / (2*bL);

        % test if block satisfy conditions for a congenstion
        % count only connected congestion, gaps are allowed.
        % gap is #gaps allowed between two 'congestion-blocks'
        gap = 1;
        if bV < 1  &&  bD > .48
            bC = bC + 1;
            bE = 0;
        elseif bE >= gap
            bC = bC - gap;
            break
        else
```

```
            bE = bE + 1;
            bC = bC + 1;
        end
        % reset variables
        bV = 0;
        bD = 0;
    end

    k = measureInterval * 60;
    congLength(1,1+mod(t-1,k)) = bC * bL;
    % measure congestion for interval[min], store mean in congPlot
    if mod(t,k) == 0
        congPlot(1,t/k) = sum(congLength/conv) / k;
        %congPlot(1,t/k) = round((sum(congLength/conv) / k)/.5)*.5;    % round half up

        % "traffic optimization" (faster congestion growth at beginning)
        if congPlot(1,t/k) > 1 && first == 1
           moveProb = moveProb + moveCorr;       % slow down moveProb when congestion begins
           first = 0;
           congStart = t;
           opt_act = 1;
        end

        % enable redlight, if congestion longer than 1km
        if congPlot(1,t/k) > 1 && redLight_act
            redLight = 1;
        else
            redLight = 0;
        end
        currentCongestion = congPlot(1,t/k);
    end

    % reset moveProb after x*3600 seconds
    if t >= congStart + 2*3600  &&  currentCongestion < 2  &&  opt_act
       moveProb = moveProb - moveCorr;
       first = 0;
       opt_act = 0;
    end

    % reset counters
    bC = 0;
    bE = 0;

    %% animation
    if isAnimated
        %clf;
        hold on;
        xlim([N-100 N+1])
        ylim([-20 20])
        plot(N-100:cC+1, 0.5*ones(1,length(N-100:cC+1)), 'Color', [.75 .75 .75], 'LineWidth', 12)
        plot(N-100:N+1, -0.5*ones(1,length(N-100:N+1)), 'Color', [.75 .75 .75], 'LineWidth', 12)
        plot(N-100:cC+1, 0*(N-100:cC+1), '--', 'Color', [.95 .95 .95], 'LineWidth', .8)
        title([ 'Iterationsschritt: ' num2str(t), '  Congestion: ' num2str(
```

```
            currentCongestion), '  Average Speed: ' num2str(vAverage), '  inFlow:
             ' num2str(rateI)])

        for row = 1:2
            for i = N-100:N
                if X(row,i) ~= -1
                    draw_car(i, 1.2*(1.5-row), 0.8, 0.2);
                end
            end
        end
        pause(0.01)
    end
end
%% END OF MAIN LOOP

disp (['inflow and outflow are equal to ' num2str(inflowCounter)]);


% diagram of comparison model - reality
[Y_dataset, data] = interpolate(dataset, measureInterval, 0);
figure
hold on;
title(' Comparison Model - Reality ')
xval = measureInterval / 60;
bar(xval:xval:nIter/3600,congPlot);
bar(xval:xval:nIter/3600,Y_dataset, .4, 'FaceColor',[.8,.85,.9])

ylim([0 16]);
if mod(dataset,2)
    xlabel([datestr(data(1,:),'dd.mm.yyyy') ' - ' datestr(data(end,:),'dd.mm.
        yyyy')]);
else
    xlabel(datestr(data(1,:),'dd.mm.yyyy'));
    if dataset == 2
       ylim([0 6]);
    end
end
ylabel('Kilometer');
if mod(dataset,2)
    xlim([0 48]);
else
    xlim([0 24]);
end
grid on;
hold off;

%% error evaluation
area_dataset = sum(Y_dataset);
error = 0;
for i = 1:length(congPlot)
    error = error + abs(congPlot(i)-Y_dataset(i));
end
error =  error / area_dataset;
if error > 1
   error = 1;
end
precision = 1 - error;

```

```
intersect = 0;
soloCounter = 0;
for i = 1:length(congPlot)
    if Y_dataset(i) > 0
        soloCounter = soloCounter + 1;
        if congPlot(i) > 0
        intersect = intersect + 1;
        end
    end
end
prediction = intersect / soloCounter;

end

%% datasets
function [InFlow, Speed] = Datasets(inSet)
switch inSet
    case 1
        InFlow = csvread('Dataset.csv', 2,   1, 'B3..B962')';
        Speed  = csvread('Dataset.csv', 2,   2, 'C3..C962')';
    case 2
        InFlow = csvread('Dataset.csv', 2,   5, 'F3..F482')';
        Speed  = csvread('Dataset.csv', 2,   6, 'G3..G482')';
    case 3
        InFlow = csvread('Dataset.csv', 2,   3, 'D3..D962')';
        Speed  = csvread('Dataset.csv', 2,   4, 'E3..E962')';
    case 4
        InFlow = csvread('Dataset.csv', 482,   5, 'F483..F962')';
        Speed  = csvread('Dataset.csv', 482,   6, 'G483..G962')';
end
end
```