Universiteit Utrecht

Utrecht University
Department of Mathematics

Master Thesis

Improving traffic mathematically: Expanding on the work of Crisostomi et al. (2011)

Author:
Björn Klaassen Bos

Supervisors:
Dr. Karma Dajani
Dr. Ross Kang

August 20, 2014

Summary

In their 2011 article, Crisostomi, Kirkland and Shorten introduced a new paradigm for modeling road network dynamics using Markov chains to process data harvested in near real time from the roads. It was shown to work, although only for a limited test case. I used the same method and compared the results with results from traffic simulator SUMO. Comparing the calculated stationary distribution, mean first passage time matrix and Kemeny constant to the results from SUMO will show us how their idea works for larger, more complicated networks as well.

Contents

Preface

The pile of paper you are holding at this very moment is my master thesis, the result of my research done over the last two years at the mathematical department of the University of Utrecht.
First and foremost I would like to thank my main supervisor, Karma Dajani, senior lecturer and researcher at the Mathematical Institute of the University of Utrecht. When I approached her in December of 2011 to ask her to be my supervisor, she immediately said yes and within a week of our first meeting I had a subject and was working on my first simulation. Even though things were not going the way I had hoped during the first half year and I was not available for the second half year, she was still there when I restarted my research in February of 2013 and has supported me perfectly along the way. Unfortunately, she was not able to supervise me until the very end due to some medical issues. Karma, thank you very much for all the time invested in me.
Second, I would like to thank Ross Kang, assistant professor at rhe Radbouw University and previously at the University of Utrecht, for taking over for Karma on very, very short notice. Along with Ross I would also like to thank Roberto Fernández for volunteering to be the second reader for my thesis.
Last, I would like to thank Jakob Erdmann, Daniel Krajzewicz and everyone else behind the sumo-user mailing list for helping me creating my simulations. From the silly questions about the tutorials when I was just starting to the slightly harder questions when I was getting closer to finishing my simulations, you were there to answer them within a two-day period, which made sure I could continue working on it as fast as possible.
To each and every one of you, thank you very much!

1 Introduction and Theory

Controlling car traffic and having good road infrastructure are important problems. The world population is getting richer, so almost every household has a car and a quarter of all households in the Netherlands have two or more (Centraal Bureau voor de Statistiek [CBS], 2012). Meanwhile governments keep expanding the roads to give all these cars a place to drive, but without too much success. Traffic jams are getting less common, but only Belgium has more traffic jams than the Netherlands in Europe (INRIX, 2012). Even with the newest traffic navigation, which can gather info on traffic jams and send you down a different route, this cannot be averted, because it will only work around a traffic jam when it is already there and cannot warn you for traffic jams which are about to appear. Also, when they start working around the traffic jam, all the navigators might send cars down the same alternative routes, which might cause traffic jams on those routes instead.
Not only is a traffic jam annoying for the people who are in it and has a negative effect on the economy, causing delays and costing the drivers gas while they are not moving, but also it has negative effects on the environment. When the cars are stuck in a traffic jam, the motors are still running, polluting the air with exhaust fumes. All in all there is a lot to win by reducing traffic jams.
One possible solution is a system which can predict traffic flows and in such a way that it can predict traffic jams and try to prevent them by proposing different routes for the drivers. If this was a possibility at this moment, a lot of time and money could be saved and we could reduce the amount of harmful emissions expelled by idle cars. This sounds quite hard to do, but with the right information and a few mathematical techniques this might not be as hard as it seems.
In this thesis I will apply Markov chains and the theory put forward by Crisostomi et al. to simulate a real-time traffic scenario just like a true traffic simulator would and show how this could help reduce traffic congestion and pollution, based on an earlier article which has showed how to do this for a very basic network.

1.1 Markov Chains

1.1.1 Random processes and Markov Chains

A random process is a process in which, at certain times, a transition takes place from one state to another. These times are set from the start, as are the states. These are gathered in the state space (X), a set of all states which the process can visit. You can distinguish between discrete and continuous variants, both for states as well as time. In our case, we will only use random processes which are discrete in both state spaces (so the states are fixed points, not intervals) and time (so we may consider $\mathbb{Z}$ as the set of possible times). This will keep both the state space and the set of times either finite or countable.
The transitions will occur at these times, among those states, according to transition probabilities which are non-negative. The process will move to any of the states within the state space (even the state it is in at the moment). So

$$p_{ij} \geq 0$$

for $i, j \in X$ and

$$\sum_{j \in X} p_{ij} = 1$$

where p_{ij} denotes the probability that the random process will make the transition to state j at the next time, given it is in state i.
Let the random variable x_t denote the state the random process is in at time t ($t \in \mathbb{N}$). A random process is called a Markov chain if it fulfils the following condition

$$\begin{aligned} P(x_{k+1} = i_{k+1} | x_k = i_k, x_{k-1} = i_{k-1}, \dots, x_0 = i_0) \\ = P(x_{k+1} = i_{k+1} | x_k = i_k) \end{aligned} \tag{1}$$

What this condition says is that the probability the process will make its transition to a certain state is only dependent on where the process is, not on where it has been in the past. The notation $P(A|B)$ is the standard notation for conditional probability and is used to express the probability of A happening, given B has happened.

Using the one-step probabilities as entries, we can construct the transition matrix P

$$P = \begin{pmatrix} p_{00} & p_{01} & \cdots & p_{0i} & \cdots \\ p_{10} & p_{11} & \cdots & p_{1i} & \cdots \\ \vdots & \vdots & & \vdots & \\ p_{i0} & p_{i1} & \cdots & p_{ii} & \cdots \\ \vdots & \vdots & & \vdots & \end{pmatrix}$$

1.1.2 Connection to graph theory

Every transition matrix can be translated to a corresponding directed, edge-weighted graph (and vice versa). This can be done for every $n \times n$ matrix by creating n nodes which represent the states $i \in X$ and directing an arc from i to j whenever the transition probability is non-zero. To give an example of this translation,

$$A = \begin{pmatrix} 0.5 & 0.25 & 0.25 \\ 0 & 0 & 1 \\ 0.5 & 0.5 & 0 \end{pmatrix} \tag{2}$$

will give us the directed, weighted graph in Figure 1.

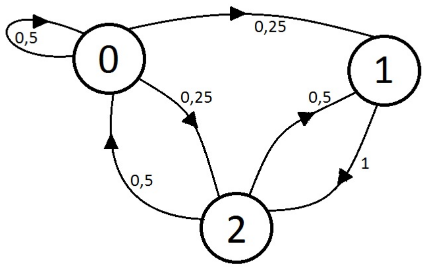


Figure 1: The graph corresponding to transition matrix A

And if we would have been given the graph first, we could use the reverse translation to acquire the transition matrix.

1.1.3 Terms used in this paper

In this paper, as in the original article, some terms and calculations are used when working with Markov chains.

Connectivity

Connectivity is one of the more commonly found terms when working with graphs and is mostly used in classifying states within the graphs. State j is *accessible* from state i when there is an n such that $P_{ij}^{(n)} > 0$, noted as $i \to j$. If the opposite is also the case (so state i is also accessible from state j) it is said that state i and state j communicate, which will be noted as $i \leftrightarrow j$. If this is true for every pair of states, the graph is said to be strongly connected. So the directed graph with state space X is strongly connected when

$$\forall i, j \in X, i \leftrightarrow j$$

If a graph is strongly connected, its corresponding transition matrix is called irreducible. We will be working with graphs and networks representing streets in an urban environment, which are meant to connect every point in the city to every other point in the city. This means that the graphs we will be working with will be strongly connected, so our matrices will be irreducible.

The left Perron eigenvector

If a graph is strongly connected (and so, the matrix is irreducible), the left Perron eigenvector π can be determined. This eigenvector, also known as the stationary distribution, can be calculated in two similar manners. For any irreducible matrix P,

1. the left Perron eigenvector is the unique vector π which will satisfy $\pi^T P = \pi^T$ where $\pi > 0$; and

2. if P is aperiodic, the stationary distribution of a matrix P can be found by solving $S = \lim\limits_{n\to\infty} P^n$ and then looking at a single row.

A matrix is aperiodic if there exists an $n \geq 1$ such that all entries of P^n are strictly positive.

Mean first passage time and group inverse

Knowing the stationary distribution of a matrix allows for new calculations to be made. Especially when it comes to finding the mean first passage time, the expected number of steps it takes to reach a certain state from a certain other state, the stationary distribution is very useful.
Given a matrix P, its group inverse $P^{\#}$ is the unique matrix satisfying

- $PP^{\#} = P^{\#}P$
- $PP^{\#}P = P$
- $P^{\#}PP^{\#} = P^{\#}$

Meyer (1975) showed many properties and applications of the group inverse and especially the information it holds on Markov chains. Ben-Israel (2008) showed how to calculate the group inverse of a matrix A. We can compute $A^{\#}$ of this square matrix by finding the factorization

$$A = CR$$

where C is equal to matrix A but without the last column (so its size is $n \times (n-1)$) and R is the identity matrix of size $(n-1) \times (n-1)$ where an extra column is added containing all -1 elements (so it has size $(n-1) \times n$). A then has group inverse $A^{\#}$, if and only if RC is nonsingular. In this case,

$$A^{\#} = C(RC)^{-2}R \tag{3}$$

Cho and Meyer (2001) showed how to use the group inverse to find the mean first passage time matrix for a transition matrix. By using the entries of the group inverse matrix of $(I - P)$, where I denotes the identity matrix of appropriate dimensions, we will denote the group inverse $(I-P)^{\#}$ as $Q^{\#}$. Using these we can find our mean first passage time matrix M by calculating its entries m_{ij}, the expected number of steps it takes to reach state j given we started in state i by using the following theorem, which is proved at the end of the section.

Theorem 1.1.1.

$$m_{ij} = \frac{q^{\#}_{jj} - q^{\#}_{ij}}{\pi_j},\ i \neq j \tag{4}$$

and

$$m_{ii} = \frac{1}{\pi_i}, i = 1, ..., n \tag{5}$$

Proposition 1.1.2.
If A^n tends to O (the zero matrix) as n tends to infinity, then $(I - A)$ has an inverse and

$$(I - A)^{-1} = I + A + A^2 + ... = \sum_{k=0}^{\infty} A^k \tag{6}$$

Proof. Consider the identity

$$(I - A)(I + A + A^2 + ... + A^{n-1}) = I - A^n \tag{7}$$

By hypothesis we know the right side tends to I. This matrix has determinant 1. Hence, for sufficiently large n, $I - A^n$ must have a non-zero determinant. But the determinant of the product of two matrices is the product of the determinants, hence $I - A$ cannot have a zero determinant. The determinant not being equal to zero is a sufficient condition for a matrix to have an inverse. Hence, $I - A$ has an inverse. Since it exists, we may multiply both sides of (7) by it

$$I + A + A^2 + ... + A^{n-1} = (I - A)^{-1}(I - A^n) \tag{8}$$

Since $I - A^n$ tends to I, the right side tends to $(I - A)^{-1}$, completing the proof. □

Definition 1.1.3
Define an absorbing state to be a state that, once entered, cannot be left. Define a Markov chain to be absorbing if there is at least one absorbing state and every other state in the Markov chain can reach an absorbing state.

Definition 1.1.4
For an absorbing Markov chain with transition matrix P we define the fundamental matrix to be $N = (I-R)^{-1}$, where R is the submatrix of P containing all non-absorbing states.

Definition 1.1.5
Define a state to be transient if, given we start in this state, there is a non-zero probability we will never return to this state. Define T to be the set containing all transient states.

Definition 1.1.6
For transient states j, We define $\mathbf{n}_j$ to be the function giving the total numbers of times that the process is in state j. We define $m_i[\mathbf{n}_j]$ to be the expected number of times that, starting in state i, the process will be in state j. We define $\mathbf{u}_j^k$ as the function that is 1 if the process is in state j after k steps and 0 otherwise.

Definition 1.1.7
Define e to be the vector with 1 for every entry and e_i to be the vector with 1 for its ith entry and 0 for all other entries.

Definition 1.1.8
Define $\{m_{ij}\}$ to be the matrix containing m_{ij} as its entries.

Proposition 1.1.9
For an absorbing Markov chain with transition matrix P,

$$\{m_i[\mathbf{n}_j]\} = N \tag{9}$$

where $i, j \in T$.

Proof. It is easily seen that $\mathbf{n}_j = \sum_{k=0}^{\infty} \mathbf{u}_j^k$

$$\begin{aligned}
\{m_i[\mathbf{n}_j]\} &= \{m_i[\sum_{k=0}^{\infty} \mathbf{u}_j^k]\} \\
&= \{\sum_{k=0}^{\infty} m_i[\mathbf{u}_j^k]\} \\
&= \{\sum_{k=0}^{\infty} ((1 - p_{ij}^{(k)}) \cdot 0 + p_{ij}^{(k)} \cdot 1\} \\
&= \sum_{k=0}^{\infty} \{p_{ij}^{(k)}\} \\
&= \sum_{k=0}^{\infty} R^k && \text{since state } i, j \text{ are transient} \\
&= (I - R)^{-1} && \text{by theorem 1.1.2} \\
&= N && \text{by definition 1.1.4}
\end{aligned}$$

Which completes the proof. □

So given the transition matrix P of our Markov chain. To find the mean first passage time matrix M, we define P_n to be P with state n absorbing, changing the nth row to e_i^T. The mean first-passage time m_{in} in P is equal to the sum of all $m_i[\mathbf{n}_j]$ in P_n for $j \neq n$. From theorem 1.1.9 it follows these are the values of row i of N, so we find the entries m_{in} to our matrix M

$$m_{in} = \sum_{j \neq n} m_i[\mathbf{n}_j] = e_i^T N e \tag{10}$$

Definition 1.1.10
A (1)-inverse X for a matrix A is the matrix fulfilling only the second group inverse criteria, meaning $AXA = A$. Meyer (1975) proved

$$\begin{pmatrix} N & 0 \\ 0 & 0 \end{pmatrix} \tag{11}$$

to be the (1)-inverse of $Q = (I - P)$. He also showed how

$$I - QQ^{\#} = \lim_{n \to \infty} P^n \tag{12}$$

Proposition 1.1.11

$$Q^{\#} = QQ^{\#} \begin{pmatrix} N & 0 \\ 0 & 0 \end{pmatrix} QQ^{\#} \tag{13}$$

Proof.

$$\begin{aligned} & QQ^{\#} \begin{pmatrix} N & 0 \\ 0 & 0 \end{pmatrix} QQ^{\#} & \\ &= Q^{\#}Q \begin{pmatrix} N & 0 \\ 0 & 0 \end{pmatrix} QQ^{\#} & \text{Group inverse property 1} \\ &= Q^{\#}QQ^{\#} & \text{(1)-inverse property} \\ &= Q^{\#} & \text{Group inverse property 3} \end{aligned}$$

Completing the proof. □

Proposition 1.1.12

$$QQ^{\#} = I - e\pi^T \tag{14}$$

Proof. Using definition 1.1.9. we see $I - QQ^{\#} = \lim_{n\to\infty} P^n$. $\lim_{n\to\infty} P^n = \Pi$, where Π is the matrix with π for all of its rows. Since $\Pi = e\pi^T$, $\lim_{n\to\infty} P^n = \Pi = e\pi^T$, so $I - QQ^{\#} = e\pi^T$. □

Proof of theorem 1.1.1. Combining (13) and (14) we get

$$Q^{\#} = (I - e\pi^T)\begin{pmatrix} N & 0 \\ 0 & 0 \end{pmatrix}(I - e\pi^T) \tag{15}$$

which, by simple matrix multiplication, gives us

$$Q^{\#} = \begin{pmatrix} (I - e\overline{\pi}^T)N(I - e\overline{\pi}^T) & -\pi_n(I - e\overline{\pi}^T)Ne \\ -\overline{\pi}^T N(I - e\overline{\pi}^T) & \pi_n\overline{\pi}^T Ne \end{pmatrix} \tag{16}$$

where $\pi^T = (\overline{\pi}^T, \pi_n)$.

By looking at the last column of (16) it is apparent that if $i \neq n$ then

$$\begin{aligned} q^{\#}_{in} &= -\pi_n\left[(I - e\overline{\pi}^T)Ne\right]_i \\ &= -\pi_n e_i^T(I - e\overline{\pi}^T)Ne \\ &= -\pi_n e_i^T Ne + \pi_n\overline{\pi}^T)Ne \end{aligned}$$

and

$$q^{\#}_{nn} = \pi_n\overline{\pi}^T Ne$$

Combining the two statements above with (10) gives us

$$q^{\#}_{in} = q^{\#}_{nn} - \pi_n m_{in} \tag{17}$$

or

$$m_{in} = \frac{q^{\#}_{in} - q^{\#}_{nn}}{\pi_n} \tag{18}$$

which proves (4) and (5) follows from this. □

Kemeny Constant

For any given transition matrix, the Kemeny constant will give you the expected number of steps needed to get from a starting state to a random destination taken from the stationary distribution. It can be calculated by

$$K = \sum_{i \in X} m_{ij} \pi_j \tag{19}$$

Kemeny and Snell (1960) showed how the Kemeny constant K is independent of the starting state i. This fact is also known as the Random Target Lemma.

Lemma 1.1.12 **Random Target Lemma**
The value K of an irreducible Markov chain with state space X, transition matrix P and stationary distribution π, computed using (19), is independent of starting state i.

For coming proofs we have to define two kinds of hitting times.

Definition 1.1.13
Define the hitting time τ_i to be the first time the random walk visits state i, so $\tau_i := min\{t \geq 0 : x_t = i\}$.
Define the first return time τ_i^+ to be the first time the random walks returns to state i, so $\tau_i^+ := min\{t \geq 1 : x_t = i\}$.
By this definition, we find $E(\tau_i^+|x_0 = i) = m_{ii} = \frac{1}{\pi_i}$, where $E(\tau_i|x_0 = i) = 0$.

Definition 1.1.14
Define a function h on X to be harmonic at i if

$$h(i) = \sum_{j \in X} p_{ij} h(j) \tag{20}$$

We call a function f defined on X to be harmonic if it is harmonic at every state $x \in X$.

Proposition 1.1.15
For each $j \in X$, the function h_j on X defined by $f_j(i) = m_{ij}$ is harmonic.

Proof. To show this, we first show how $E(\tau_j|x_0 = i)$ is harmonic. For notational convenience, let $h_j(i) = E(\tau_j|x_0 = i)$, then $\sum\limits_{k\in X} h_j(k)p_{ik} = (Ph_j)(i)$. Observe that if $i \neq j$,

$$\begin{aligned}
h_j(i) &= E(\tau_j|x_0 = i)\\
&= \sum_{k\in X} E(\tau_j|x_1 = k)p_{ik}\\
&= \sum_{k\in X} (1 + m_{kj})p_{ik}\\
&= \sum_{k\in X} (1 + h_j(k))p_{ik}\\
&= \sum_{k\in X} p_{ik} + h_j(k)p_{ik}\\
&= 1 + \sum_{k\in X} h_j(k)p_{ik}\\
&= 1 + (Ph_j)(i)
\end{aligned}$$

so that

$$(Ph_j)(i) = h_j(i) - 1 \tag{21}$$

If $i = j$, then,

$$\begin{aligned}
E(\tau_i^+|x_0 = i) &= \sum_{k\in X} E(\tau_i^+|x_0 = i, x_1 = k)\\
&= \sum_{k\in X} (1 + h_i(k))p_{ik}\\
&= \sum_{k\in X} p_{ik} + h_i(k)p_{ik}\\
&= 1 + \sum_{k\in X} h_i(k)p_{ik}\\
&= 1 + (Ph_i)(i)
\end{aligned}$$

Since $E(\tau_i^+|x_0 = i) = m_{ii}$, (5) shows $(Ph_i)(i) = \frac{1}{\pi_i} - 1$,

$$\sum_{k\in X} h_i(k)p_{ik} = \frac{1}{\pi_i} - 1 \tag{22}$$

Thus, letting $h(i) := \sum_{j \in X} h_j(i)\pi_j$, (21) and (22) show that

$$\begin{aligned}
(Ph)(i) &= \sum_{j \in X} (Ph_j)(i)\pi_j \\
&= \sum_{j \neq i} (h_j(i) - 1)\pi_j + (\tfrac{1}{\pi_i} - 1)\pi_i \\
&= \sum_{j \neq i} h_j(i)\pi_j - \pi_j + 1 - \pi_i \\
&= \sum_{j \neq i} h_j(i)\pi_j \\
&= \sum_{j \in X} h_j(i)\pi_j - h_i(i)\pi_i \\
&= \sum_{j \in X} h_j(i)\pi_j \\
&= h(i)
\end{aligned}$$

Fulfilling the definition of being harmonic.
Now, since $h_j(i) = E(\tau_j | x_0 = i)$ is harmonic, we can expand this to our formula of K.

$$\begin{aligned}
K &= \sum_{i \in X} m_{ij}\pi_j \\
&= \sum_{j \neq i} m_{ij}\pi_j + m_{ii}\pi_i \\
&= 1 + \sum_{j \neq i} m_{ij}\pi_j \\
&= 1 + \sum_{j \neq i} E(\tau_j | x_0 = i)\pi_j \\
&= 1 + \sum_{j \in X} E(\tau_j | x_0 = i)\pi_j - E(\tau_i | x_0 = i)\pi_i \\
&= 1 + \sum_{j \in X} E(\tau_j | x_0 = i)\pi_j
\end{aligned}$$

Thus, since $\sum_{j \in X} E(\tau_j | x_0 = i)\pi_j$ does not depend on i, K also does not depend on i.

$\square$

Proposition 1.1.16
If the transition matrix P of a Markov chain is irreducible with state space X, a function h which is harmonic at every point of X is constant.

Proof. Since X is finite, there exists a state x s.t. $h(x) = b$ is maximal. If for some state z s.t. $p_{xz} > 0$ we have $h(z) < b$, then since h is harmonic and $\sum\limits_{y \in X} p_{xy} = 1$

$$h(x) = \sum_{y \in X} h(y) p_{xy} = p_{xz} h(z) + \sum_{y \neq z} p_{xy} h(y) < b, \tag{23}$$

a contradiction. Thus, $h(z) = b$ for all states z s.t. $p_{xz} > 0$.
For any $y \in X$, irreducibility implies a sequence $x, x_1, ..., x_n = y$ with $p_{x_i, x_{i+1}} > 0$. Repeating above argument tells us $h(y) = h(x_{n-1}) = ... = h(x) = b$. Thus, h is constant.

□

Proof of Lemma 1.1.12. Combining 1.1.15 and 1.1.16, we can conclude the function for K is constant and therefore does not depend on starting state i

□

1.2 SUMO

In both the original article and my extension of their work, a traffic simulator was needed to collect information and to compare results. In the original article, the authors chose to use SUMO and so will I.
Simulation of Urban MObility, or SUMO (Behrisch et al. 2011) is an open source, highly portable, microscopic, multi-modal road traffic simulation package developed by the Institute of Transportation Systems of the DLR, the German Center for Aerospace Travelling. It is designed to handle everything from simple to large road networks. Although SUMO is primarily created for use on Windows, there are Linux packages available and since it is open source, you are welcome to alter the software in any way you like.
After downloading the SUMO package, the only other thing you need to start using SUMO is a text editor which support the .xml extension (so even Notepad and Wordpad suffice). Every file SUMO will read to create your network, or create for you, will (have to) be in the .xml extension.
To create a network and traffic flows in SUMO, you will have to create certain files first. In Figure 2 you can find a normal filetree for a SUMO simulation.

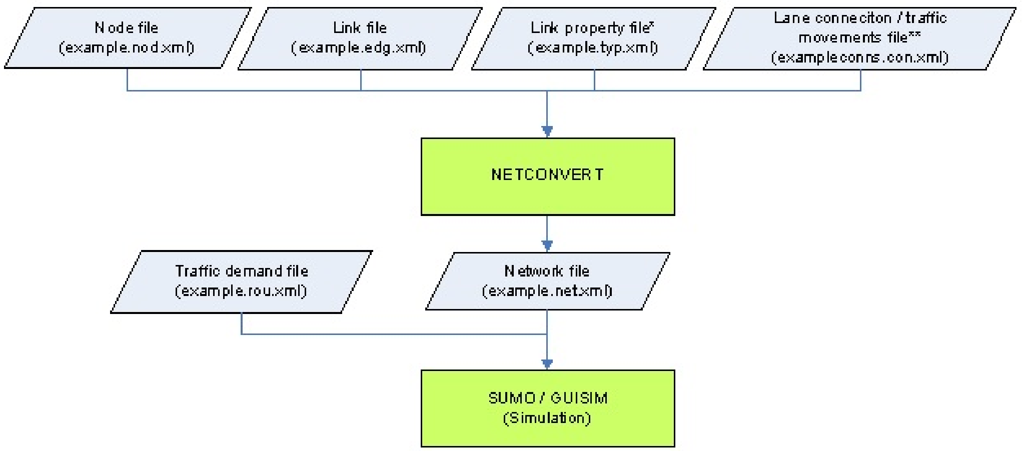


Figure 2: The filetree for creating a simulation in SUMO

1.2.1 SUMO files

There are three files which you need to make to create a simulation: A node- and edge-file (to create a network in the network-file) and a route file, to define the flow of traffic in the simulation. But these will only create a very basic simulation and if you want to simulate something more lifelike, you will have to add more files and more information. I will go through the files I used for my simulation and explain what they are used for.

Node-file (.nod.xml)

The node-file defines the location of junctions, corners in the road and other places of interest for your simulation. Nodes are defined by coordinates (x and y) where the distance to the origin is equal to the length in meters. An example of a line of the node-file (which defines one of the nodes) is:

```
<node id=”1” x=”5.0” y=”600.0” />
```

The id of the node is just a name to which it will be referred to. The coordinates and the id are the only properties which can be assigned to a node.

Edge-file (.edg.xml)

The edge-file defines the roads in your network, connecting the nodes defined in the node-file to each other. Edges are defined by a ’from’ and a ’to’ node, which means that if a road between node A and node B can be driven in both directions, the edge has to be defined twice: Once from A to B and once from B to A. An example:

```
<edge id=”SorbonnelaanS2” from=”1” to=”2” type=”b” />
```

Edges can also be given more information, like the number of lanes and maximum speed. But instead of doing this for each lane individually, it is easier to define types in the type-file.

Type-file (.typ.xml)

The type-file defines properties of the roads which are used in the network. In my case, I have defined three properties: Maximum speed (in m/s), priority (if two roads meet at a junction, the road with the higher priority will have priority) and the number of lanes. An example of a line of the type-file:

```
<type id=”b” priority=”6” numLanes=”1” speed=”13.889” />
```

Connections-file (.con.xml)

The last file before creating the network is the connections file. When a car reaches a junction, most of the time it has multiple possibilities of where it can go. On roads with multiple lanes it might even depend on the lane you are on now to which lanes of which roads you may travel. In the connection files you define all these connections, giving an edge of departure including the lane you will be departing from and an edge of arrival including the lane you will be arriving on. A line from this file could look like:

```
<connection from="SorbonnelaanS2" to="SorbonnelaanS3"
fromLane="0" toLane="0" />
```

Network configuration-file (.netc.cfg) and network-file (net.xml)

The network configuration-file is a file in which you tell SUMO which files in the current folder will give the right information about nodes, edges, etc. Using this file netconvert, a command line application which is supplied alongside SUMO, will create the network file for you, which you can use for the rest of your simulation.

Route-file (.rou.xml)

In a simple simulation, your route file consists of a lot of information. In this file, you define the type of vehicles which will drive through your network, the routes they will take and how many vehicles will be driving. The type of vehicles are defined by their respective acceleration speed, deceleration speed, ID, length of the car, maximum speed and more. The route is straightforward, the chain of edges which are part of the route and an ID by which the route can be called. The vehicles themselves are then defined by a departure time, an ID, the route they will take and the type of vehicle they are. Examples of the different lines:

```
<vType accel="3.0" decel="6.0" id="CarA" length="5.0"
    minGap="2.5" maxSpeed="50.0" />
<route id="route01" edges="D2 L2 L12 L10 L7 D7" />
<vehicle depart="54000" id="veh0" route="route01" type="CarA" />
```

This is the standard build for a route file, but this is possibly the most customizable file in your simulation. Depending on the kind of simulation you want, it is possible to have your cars take random trips, to route for the shortest or the optimal path, route from observation points or just random routes. In my scenario, we will be working with routing by turning probabilities, which means there are other files to be made first.

Turn ratio-file (.turns.xml)

If you want to route according to turning probabilities, you will have to define these probabilities for every road at every junction. This is done in the turn ratio-file, where for each edge the possibilities are given (just like in the connection file) alongside the probabilities someone will take that choice. An example:

```
<fromEdge id=”SorbonnelaanS2”>
   <toEdge id=”SorbonnelaanS3” probability=”0.05” />
   <toEdge id=”AarhuslaanE” probability=”0.95” />
</fromEdge>
```

Turn-ratios can be defined for certain intervals so you can simulate the different behavior of traffic at different times in the same simulation.

Flow-file (.flows.xml)

To complement the turn-ratio file, there will have to be some cars driving across the network. This is where the flow-file comes in. The flow files defines, per given interval, how many cars will start at a certain road and lets them depart, uniformly distributed, over that interval. This means flows will have to be defined by an ID, a starting edge and the number of cars to depart, per interval you want to define. An example:

```
<interval begin=”30” end=”60”>
   <flow id=”200” from=”SorbonnelaanS2” number=”2” />
<interval>
```

SUMO configuration-file (.sumo.cfg)

Just like the netconvert configuration-file, the SUMO configuration-file lets you sum up which file contains which information so SUMO can use them to start the simulation. You can call on this file through the command line, which will execute the simulation, or by using the SUMO-GUI, which will give you a visualization of the simulation and shows you what is happening during the simulation to find any flaws in your files.

Additional-files (.add.xml)

SUMO is very customizable and it shows in the number of additional files you can use. Adding traffic lights, public transport or variable speed signs are all possibilities within the current program. Additional files also have to be used to get any output from your simulation, whether it is only for getting the average speed the cars drove on every edge, or whether you want the full report on capacity, speed, exhaust fume levels etc.

2 The original article

Although innovative, the idea behind this thesis is not original. The idea was first published by Crisostomi, Kirkland and Shorten (2011) in their paper, *A Google-like model of road network dynamics and its application to regulation and control*. Their idea was to use cars as sensors, by equipping them with the hardware to collect information in real time and then send it to a central point, where the information can be collected and used to model and engineer road networks. The data is then processed and sent back to the cars, which can use it to plan the optimal route for the driver to take. This would allow for faster travel times, fewer traffic jams and optimal use of gasoline, which in turn will reduce the expulsion of exhaust fumes.
For the calculations, they proposed to make use of Markov chains similar to how Google's PageRank algorithm uses it to model congestion in the Internet. Markov chains offer some major advantages over commonly used road network simulators, since they are easily built from the information we can harvest by using cars as sensors, the calculations can be made quite quickly when a decent computer is used and they are capable of giving more insight into the road network.
The idea of their article was to show how Markov chains could be used to use the information you get (in the future this would be from cars, but in their case they got their information from the SUMO traffic simulator) and transform that into whatever you want to know about the network.

2.1 The need for a new method

To reduce congestion and harmful emissions, you need to be able to manage traffic flows in an intelligent manner. The key to developing strategies to manage traffic is creating traffic models which are both accurate and easy to use for predicting and controlling traffic. The biggest objective when developing models like these is finding a way to develop smart traffic management systems which proactively take pre-emptive measures to avoid incidents and traffic jams, rather than reacting to them. Another requirement is that these models should be constructed from real data obtained directly from real life situations on the network in near real time.
In the past, stochastic models have been used to simulate road networks because of how simple they are to use. An example of this is the Constant Speed Motion model (Fiore and Härri, 2008) but, as the title suggests, they only work with constant speeds, which means they only simulate very basic models. For these situations, flow models were introduced, which provide a more realistic way of modeling urban networks. The problem with flow models is that you have to define a path for every single vehicle in the flow to follow. For small networks, this is not a problem, but when cities are defined by a network of thousands of roads, the amount of work rises dramatically.
The most popular way of investigating the behavior and efficiency of road networks is using mobility simulators, which uses both stochastic models and flow models. But they still suffer from scalability issues. That is why there is a need for a new method. Using cars to acquire the information needed and then sending it to a computer which turns the information into a Markov chain to use in your simulation means you still have the simplicity of the stochastic models, while lessening the problems of scalability since the computer can easily create the matrix for you.

2.2 The network

For the example they used in their tests and the validation of their ideas, they used a very basic route network based on the following transition matrix:

$$B = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0.45 & 0.45 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0.45 & 0.45 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \end{pmatrix}$$

which corresponds to the following graph

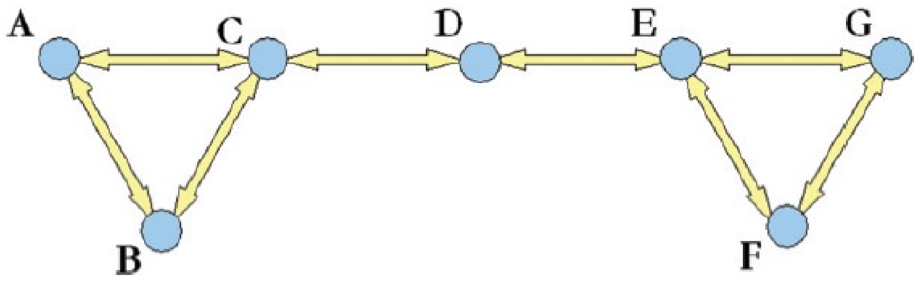


Figure 3: The graph corresponding to transition matrix B

2.2.1 Primal vs. dual

The graph as seen in Figure 3 is an example of a simple network displayed in the 'normal' way. This representation, where the nodes correspond to the junctions in the network and the edges correspond to the connecting roads, is called the primal representation. Although this is the most common way of displaying graphs, it is quite bare. It is very simple, which is great for basic calculations, but might not contain full information.
For this reason they used a different representation of graphs, namely the dual representation. In this representation, the roles are reversed and the edges represent the junctions and the nodes correspond to the streets. In Figure 4 we have a dual presentation of a network which would have the primal representation of Figure 3. The node AB corresponds to the lane connecting junction A and junction B in the original network.

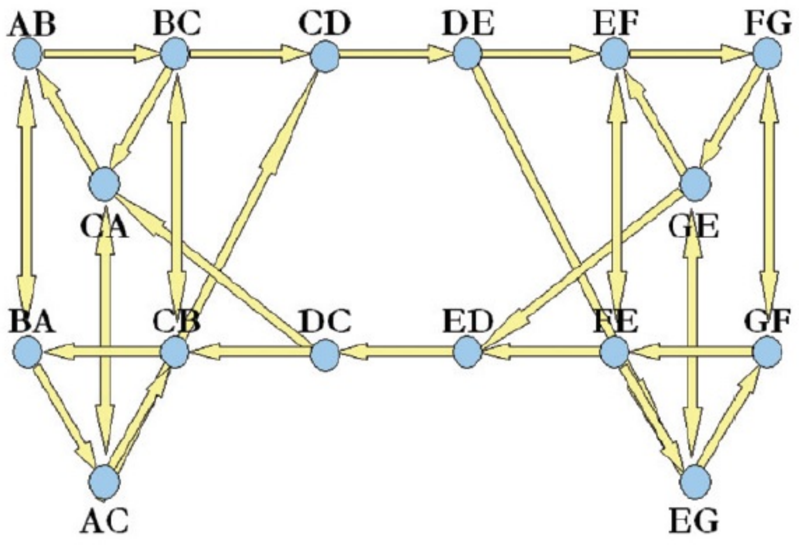


Figure 4: A dual representation of the graph shown in Figure 3

- In the primal network of Figure 3, it seems possible to drive from node E to node D, make a U-turn and return to E, while this is not possible. In the dual representation this is evident by road ED not having the possibility to reach road ED, showing how turning around is not a possibility.
- The standard way of creating traffic flows is by using junction turning probabilities. Since these depend on the road of origin (and not just the junction you are at) this is only possible in the dual representation, since this information is not available in the primal graph.
- Information like speed limits and street lengths can be incorporated into the transition matrix, to be shown later.

For these reasons, they worked with the dual representation of the graph in their article. We transform the transition matrix to

$$C = \begin{pmatrix}
0 & 0 & 0.1 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.1 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.8 & 0.1 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.9 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.9 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.9 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.1 & 0.8 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0.1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0.1 & 0 & 0
\end{pmatrix}$$

based on the original transition probabilities and turnarounds (where possible) having a probability of 0.1.

2.2.2 Calculations in more complicated situations

In this case, their simple simulation with this new transition matrix would suffice. The fact that every road is equal in length and has the same speed limit means every street can be traversed in exactly the same time. This is a very good property to have when translating the situation to a Markov chain, since Markov chains calculate steps instead of time. In this case, every step is as long as the time it takes to traverse an arbitrary road. But in real-life situations, it is very easy to find roads which do not take the same amount of time to traverse. The biggest factors deciding the average travel time for a street are speed limits (two streets of equal length but different speed limits have different travel times) and street lengths (streets with the same speed limit but different lengths will have different travel times as well). In reality there are even more factors, like the presence of traffic lights, pedestrian crossings and bus stops, the quality of the road surface, etc. So in a more complex situation, this will have to be accounted for when translating the information into a Markov chain.
The answer for this problem is pretty simple. Say we are currently on a road AC and approaching the junction which can lead us back, to road CA, or lets us move on to either road CB or CD. The probabilities of turning onto a particular road are given as

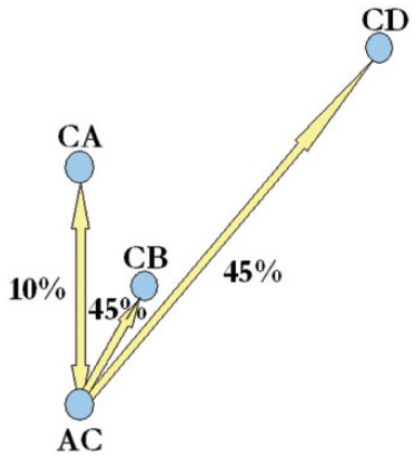


Figure 5: Turning probabilities for the junction at the end of road AC

Suppose we now replace road AC by road A'C'. Road A'C' is twice as long as road AC and also has half the speed limit of road AC. These two changes mean that, on average, the time it takes to travel across road A'C' will be four times greater than the time needed to cross road AC. To take this extra travel time into account, we will add a self-loop to road A'C' and adjust the other turning probabilities (since the total will still have to be 1).

Although it will be very hard to calculate the exact travel times using only the length of the street and the speed limit, we are working from a situation in which the cars will tell us how long it took them to traverse the road, so the travel time for every road is known. If all the travel times are then normalized (the shortest travel time will be adjusted to 1 and all the other travel times are adjusted accordingly) then we can calculate the probability value on each loop by using

$$p_{ii} = \frac{tt_i - 1}{tt_i},\ i = 1, ..., n \tag{24}$$

where tt_i is the average travel time for the i-th road, which we have collected from the data given to us by the sensors in the cars. All the other probabilities will be adjusted by dividing them by tt_i. In the case of Figure 5, this will give us

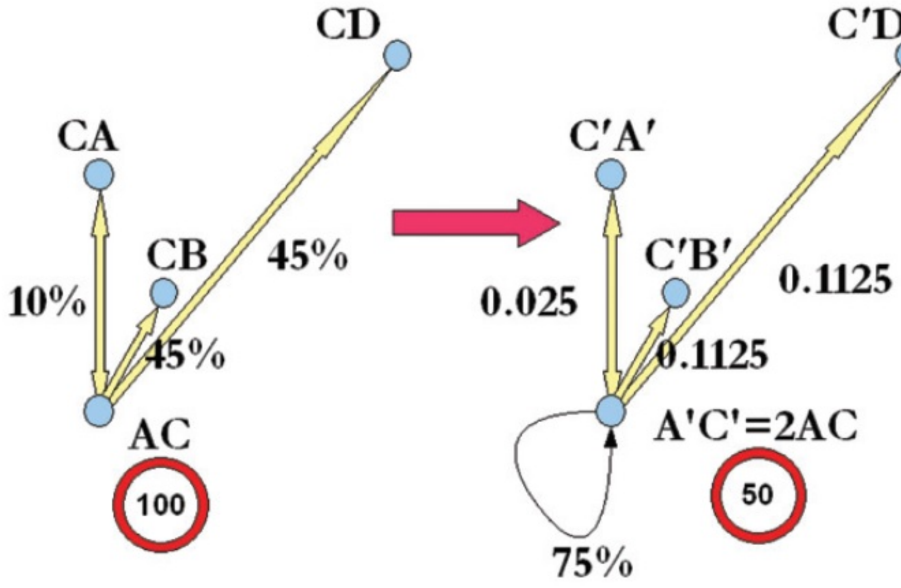


Figure 6: Turning probabilities for the junction at the end of the adjusted road A'C'

2.3 Validation

When you have ideas like this, you will have to demonstrate their validity. So, for their basic graph as given in figure 3 (and the dual representation of the graph from figure 4), Crisostomi et al. started collecting results and showed how accurate these were.

2.3.1 Stationary distribution

In a Markov chain, the stationary distribution corresponds to the long-run fraction of time the chain resides in a particular state. For road network, this means it can tell us the long-run fraction of time cars are driving across a certain road. It will not tell anything about the amount of traffic actually on it, but its information can be used to evaluate whether traffic is balanced (or if some roads get used a lot which can lead to traffic jams) and which roads are crucial for traffic flows. To get this information from SUMO, they used the output concerning the occupancy of each road (in vehicles/kilometres) from which they computed the relative density. For the Markov approach, they just found the stationary distribution of matrix C. Compared to each other we get

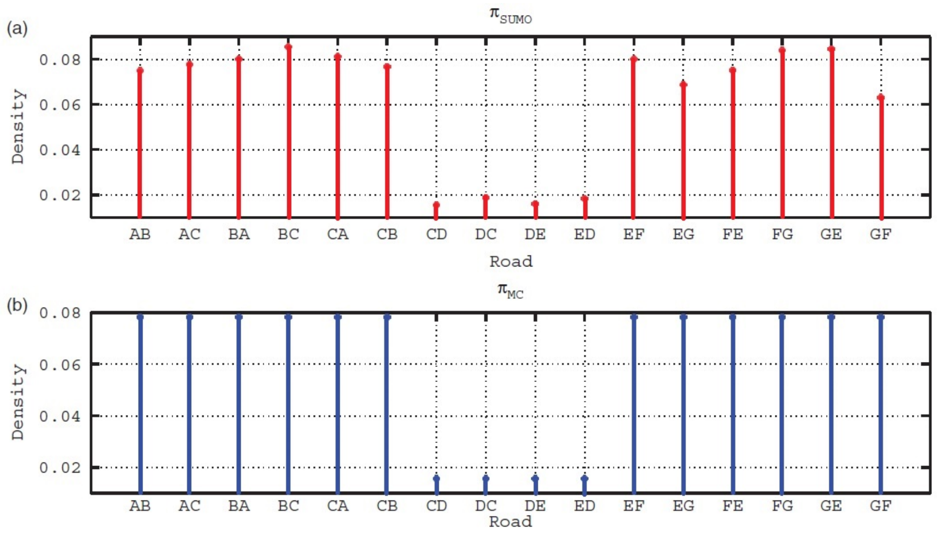


Figure 7: Comparison between the stationary distribution of cars estimated from the Markov chain approach (b) and computed from the SUMO simulation (a)

Which seems to be pretty accurate. We can also see that most cars seem to stay within the sets of junctions A-B-C or E-F-G and rarely travel to the other set.

2.3.2 Mean first passage time

The mean first passage time, although most drivers do not drive around in a random fashion until they happen to stumble upon their destination, can give us a reasonable idea of on the travel times from one state to another. For the Markov chain approach, the mean first passage time can be easily computed using the group inverse and the formulas shown in section 1.1.3. It is quite a bit harder to get this information from SUMO. For every entry of the mean first passage time matrix M a simulation was performed where a flow of cars started from the origin road until they reached the destination road. Out of all these flows, the average time was computed. The data was then displayed in both a 3D plot and a contour plot

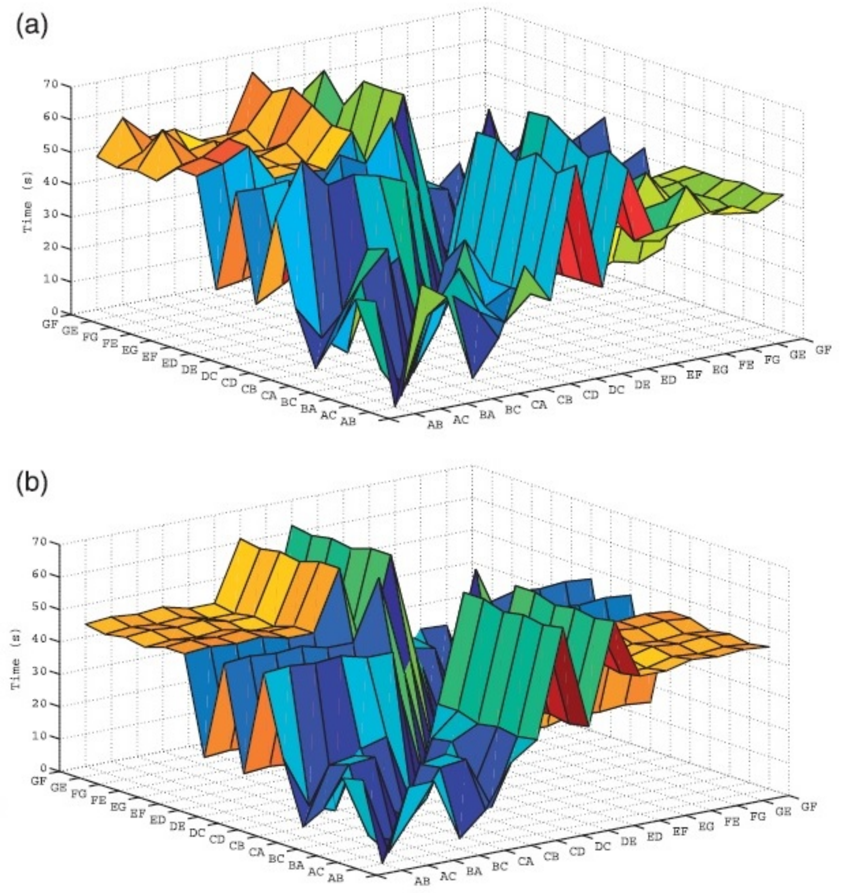
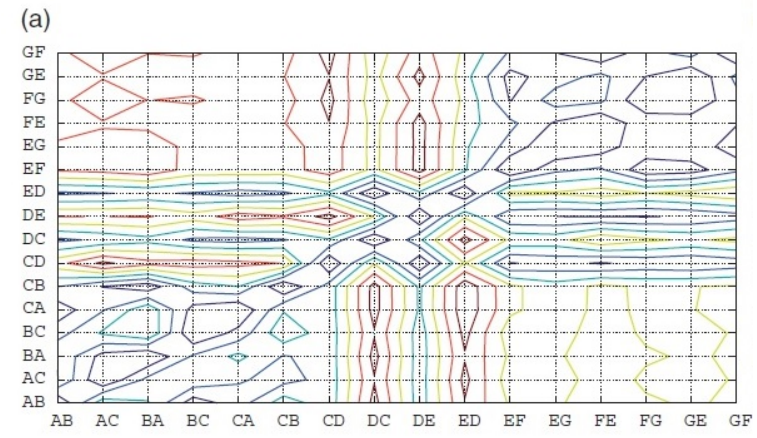


Figure 8: 3D plots of the mean first passage times extracted through different simulations in SUMO (a) and computed using the group inverse (b). The x- and y-axis contain the origin and destination roads, the z-axis shows the mean first passage time



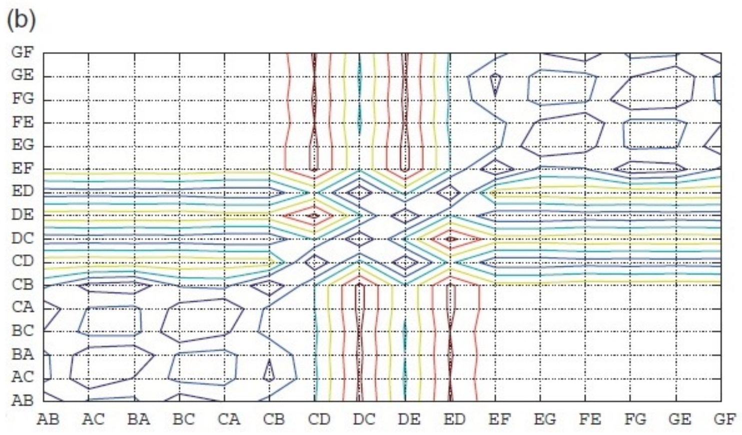


Figure 9: Contour lines of the mean first passage times from both SUMO (a) and the matrix calculation (b)

2.3.3 Kemeny Constant

Calculating the Kemeny constant from the simulation in SUMO will come down to using the same calculation on the mean first passage time matrix built from the simulation as you would on the mean first passage time gotten through the mathematical calculations. Since the 3D plots in the last section showed these were quite accurate, there is not much use in comparing the two different values for the Kemeny constant. Because of this they decided to check the other property of the Kemeny constant: it should be constant. They took the mean first passage time which they derived from the output of the SUMO simulation and used the formula from (5) to compute the Kemeny constant for every road in the network. The results were

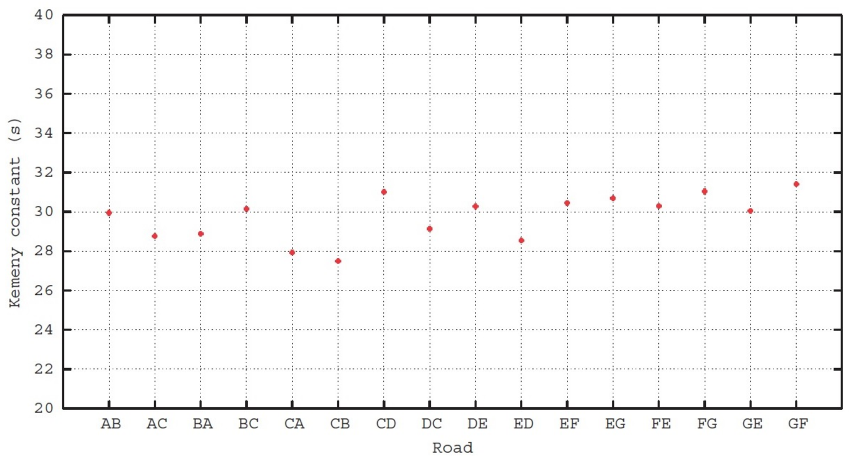


Figure 10: The Kemeny constant calculated for every starting road

As can be seen, the values are not exactly the same, but they are approximately constant, once again showing the method to be reasonably effective.

3 Expanding the test

The concept, as brought up by Crisostomi et al, seems like it can be very useful. But, if we actually were to use it for the network of roads, even if it were only in the Netherlands, the scale of the entire project would have to be enlarged greatly. Will it still work then? So far, all they have shown is how it seems to work in a very small network of seven nodes and eight edges, but most residential districts are larger then that, let alone entire towns or cities. And if one city is too much for this concept, how would you ever use it for an entire country, or even the world? Also, they only used very simple roads, which only had one lane and all had the same length and speed limit. In this part of my thesis, I will show you whether the concept will still work when you enlarge the scale of the network and add more variables to the roads itself. Different lengths, speed limits, amount of lanes and priority regulations all play a part in everyday road networks, so I will be taking them into account during my research.
In this chapter I will show you every step of the way. From deciding the real life network I would be using, deciding on nodes and edges, translating it to SUMO files for the simulation and calculating everything using the output from the simulation and MATLAB. Most of the results of what I have done can be found in the Appendix, since the files and matrices are too large to just mention in the thesis itself.

3.1 The network

3.1.1 Deciding on an area

The original test seemed to be a success, but the thing it misses the most in my eyes was realism. It will be very hard to find places in this world where the roads are of exactly equal length, with precisely the same speed limit, no traffic lights, no pedestrian crossings, etc. That is why the biggest criteria I wanted to set to the network I would be using was that it had to be a real network. Next to that, it had to be/have:

- somewhat large, since I wanted to show how the idea would work for larger networks;
- not too large, since it would mean far to much work trying to set up the simulation and the calculations;
- roads of different length;
- different speed limits;
- different numbers of lanes per road.

Next to all that, I would like it to be somewhere which is close to me. I did not take long to find two good candidates: The neighborhood I live in, or 'De Uithof', the district in the city of Utrecht where I study. After comparing the two areas, De Uithof seemed to be easier to simulate because the roads are aligned in a rectangular fashion, which made it easier to estimate the coordinates of all the different nodes I would be using.

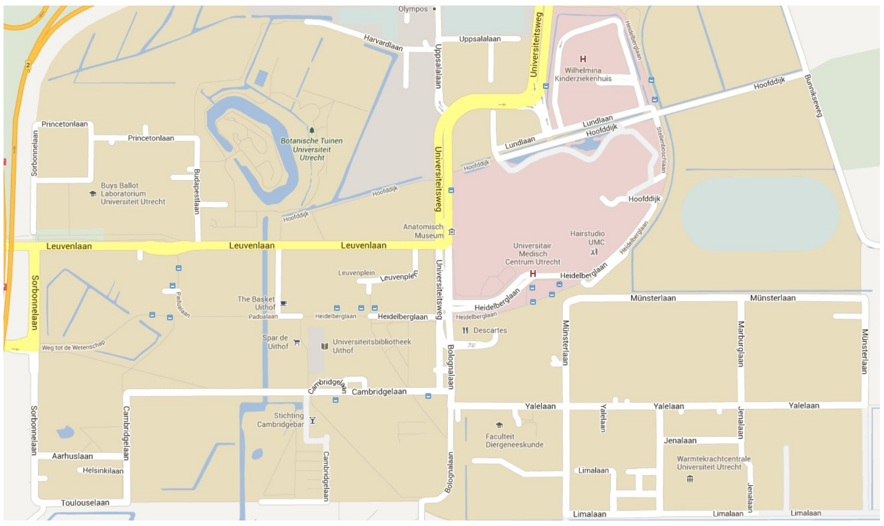


Figure 11: De Uithof by Google Maps

3.1.2 Translating real life to a network

First, I had to work on the basic elements of the network, namely the nodes and edges. Every junction which gave the driver of a car a choice between destination roads was made into a node, just as every corner in a road, since that would simplify creating the simulation. In the end, the result of naming al the nodes was the map found in figure 13, where nodes 34 and 35 are a bit more complicated than they look at first glance, as seen in figure 12

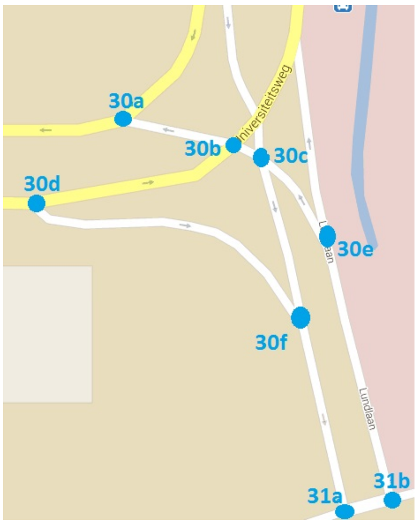


Figure 12: Zoomed in at node 34 and 35

Now, to be able to create the skeleton of the network for SUMO, I had to give every node coordinates. Since the nodes 13, 14 and 15 were the most southwards orientated nodes, I set them on the coordinates (x,0) and due to a mistake I had made when I started creating my simulation, the most westward oriented nodes 1, 2, 3, 42 and 43 were not set at (0,y) but on (5,y). Since this would not hurt the rest of the simulation, I did not bother to change the x-coordinate of every node to make up for this, so I just let them be.

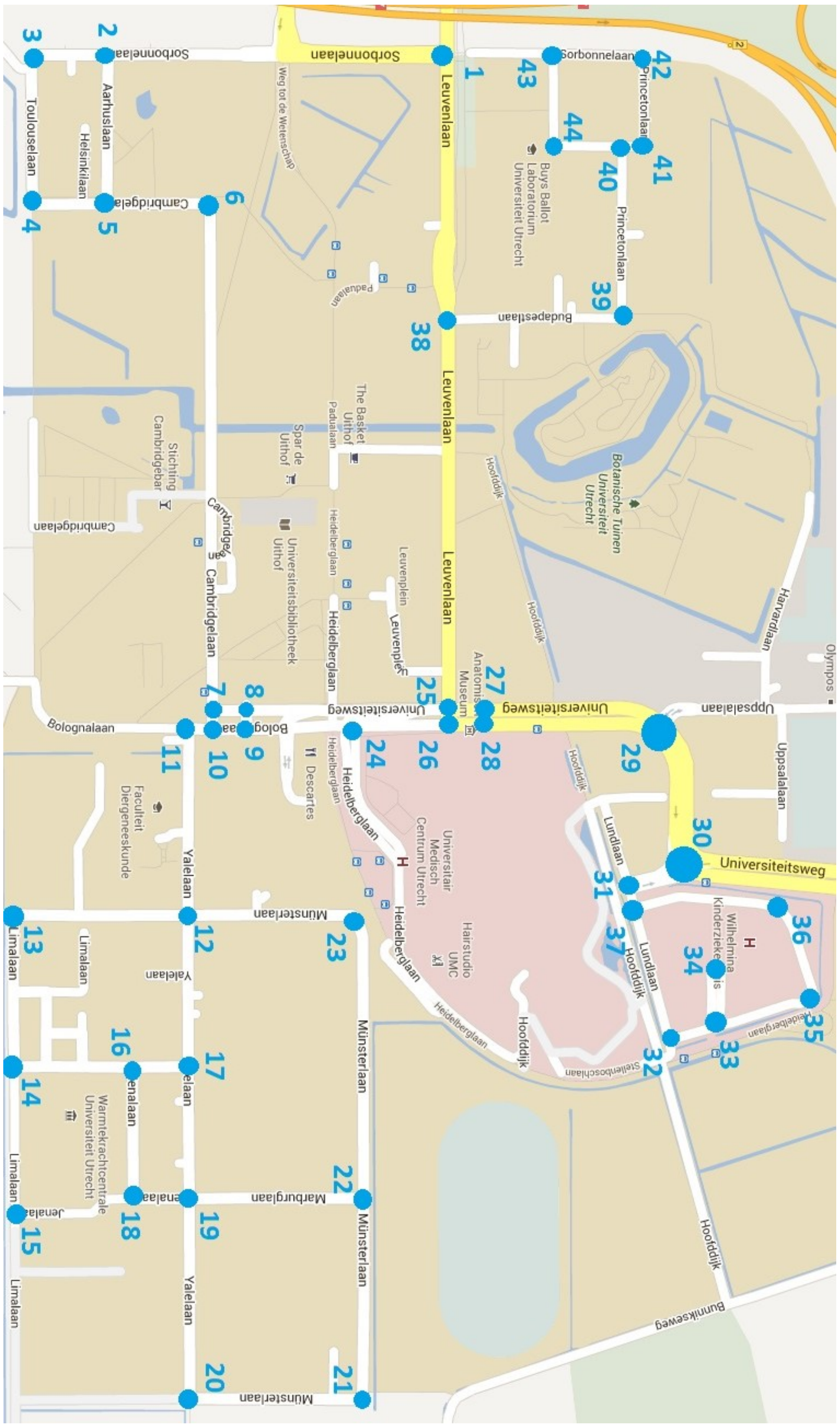


Figure 13: De Uithof in nodes

3.1.3 Creating SUMO files

Now that my origin was set at (5,0), I started finding coordinates for the other nodes. Due to the rectangular lay-out of the area it was relatively easy to do this; If I knew node 1 to be at coordinates (5,600), node 2, which lies directly to the south and according to Google Maps at 479 meters away, had to be on coordinates (5,121). Continuing along this way and experimenting a bit with the roads which were not lying in a rectangular fashion, I got the coordinates as given in the node file, which you can find in the Appendix. Although the coordinates will not all be spot on compared to the actual road network on De Uithof, it will be good enough for my simulation. Luckily, it does not have to be an exact copy, since the main reason I am using a real situation is for the different properties a road can have, not to use the exact road lengths.
Once the nodes were all set, I continued to the other basic element of a network, namely edges, or in this case, roads. For every two nodes which are connected on De Uithof, I defined an edge in the edge file. The edges were named after their actual street name, the direction the street was headed and a number if there were more edges with the same street name and direction. These numbers were given in normal reading order, either from top to bottom or from left to right. Also, for every edge some properties were defined: the number of lanes, maximum speed and a number concerning priority.
Lastly, before I let SUMO use the files I have created so far to construct the network, I had to create the connections file. Most of this was straight forwards, especially on road segments where the driver does not have a choice when reaching a node. But for every lane of every road heading towards a node, the possible directions have to be defined.

Once these files are created, SUMO can use them to compile the network, which came out like this:

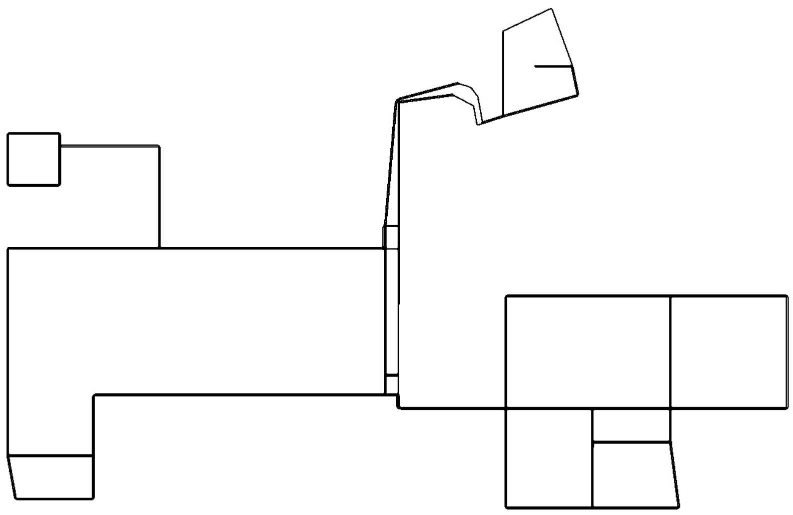


Figure 14: The network of De Uithof created by SUMO

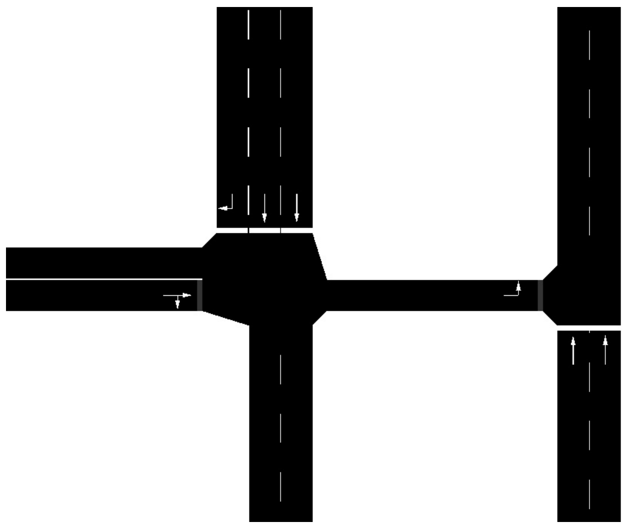


Figure 15: Close up of nodes 25 and 26 of the network

which, compared to map of figure 12, seems quite similar.

After I had created the network, all that was left for me to do was to get some cars to drive over it. Because the original article noted turning probabilities at every junction as a part of the information the sensors inside the cars could collect for them, I assumed they based their simulation on a random model where every junction has its turning probabilities and so did I. It looked like they used equal turning probabilities and a probability of 0.1 for turnarounds. I chose to have custom turning probabilities for every junction by estimating the real values. For example, turning probabilities from road A to road B would be high if getting to B would actually get them somewhere they might want to go, or would be low if it was a very illogical turnaround or route option.
To let cars drive over the network I had created, I had to create a route file. In this case, this meant creating two other files: a turning probability file, which tells SUMO if a car approaches a junction from a certain road, which directions it can take (which was also defined in the connections file) but also what the probability is it will actually take that road. The other file is a flow file, which tells SUMO where the cars will actually start. Since the network I have created is a closed network, I've chosen to only let cars spawn in the first 60 seconds, so as not to clog up the network over time. The number of cars starting at a certain road was based on the amount of time cars needed to cross the road. This was done in such a way that, after 60 seconds, the cars were almost uniformly distributed across the different roads. After this starting period, the cars would drive around randomly, although still obliged to follow the given turning probabilities.
From these two files, a route file could be created which was the last step into getting a simulation started. Now that all the files are done, it was time to have the cars drive around the network.

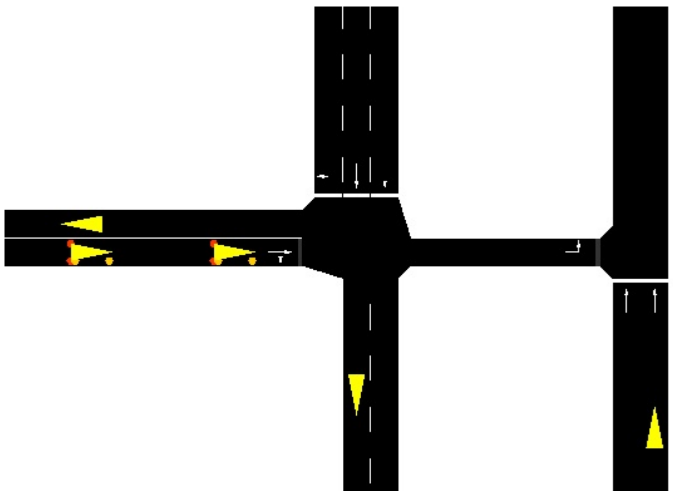


Figure 16: Cars driving at the network around node 25 and 26

3.2 Calculations and results

With the simulation created, it was time to look at the results. Before I could do this, I had to make the matrices for the mathematical approach. From the turning probabilities as mentioned in the turns file in the Appendix I created matrix T following the method mentioned in paragraph 2.2.2. Now I had the base for both the SUMO calculations and the mathematical calculations, which allowed me to continue.
For every calculation I will explain what was done to gather the results from both SUMO and from the mathematical calculations.

3.2.1 Stationary distribution

For the mathematical calculations, I used the method I explained in paragraph 1.1.3. For this, I used matrix T, the transition matrix which can be found in the Appendix. Using MATLAB, it was easy to calculate the limit to find the stationary distribution π_{MC}.
Using SUMO, this was a little more complicated. SUMO has no option to give the stationary distribution in the output, but this can be calculated otherwise. The output gave me the sampledSeconds output, which gave me the number of seconds a vehicle was present on the edge. I divided this value by the period, which was 3600 in this case, to get the average number of vehicles on the edge at any given moment. If we divide this value by the sum of the number of vehicles on all the edges, we can get the stationary distribution. The exact values can found in the appendix.

Figure 17 compares the values we found through both SUMO and the Markov chain calculations and shows the difference between the two values.

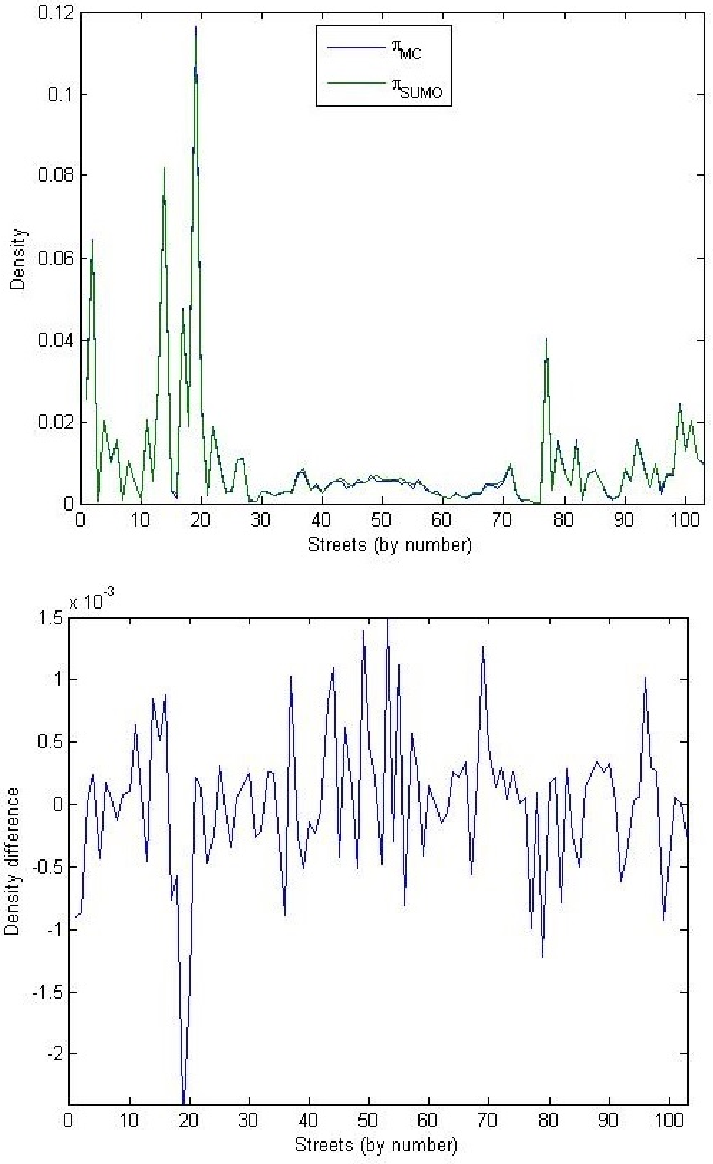


Figure 17: Comparing the stationary distributions (top) and the difference between the values (bottom).

As one can see, the values of the two stationary distributions seem to lie very close to each other and the right graph shows that the difference between the two values is, in most cases, very close to zero. For just a few streets, the values seems to differ more then 0.001 and no more then 0.0025, which means a difference of just over 2%. All in all, the approach seems to be accurate.

3.2.2 Mean first passage time

For the mathematical calculations, I used the method I explained in paragraph 1.1.3. Using matrix T to create matrix $I-T$ and then finding matrices C and R as mentioned, I used MATLAB to calculate $Q^{\#}$. This I used to calculate the mean first passage step matrix. Since the matrix works in steps and I had to normalize the values equalizing 1 step to 0.61 seconds, I had to multiply every entry of this matrix by 0.61 to get the mean first passage time matrix M_{MC}. $Q^{\#}$ and M_{MC} can be found in the Appendix.
Using SUMO, this was a lot harder. Once again, SUMO has no option to give the mean first passage time in the output. Calculating it takes a simulation for every entry of the matrix M, where I used the network and turns file of the original matrix, and altered the flow file for every simulation. In this way, I could let a flow of cars start at a certain road and let them drive around until they reached the desired destination road. In the output, I could find the time needed for every car and, in this way, calculate the average time the cars needed to reached the desired state.

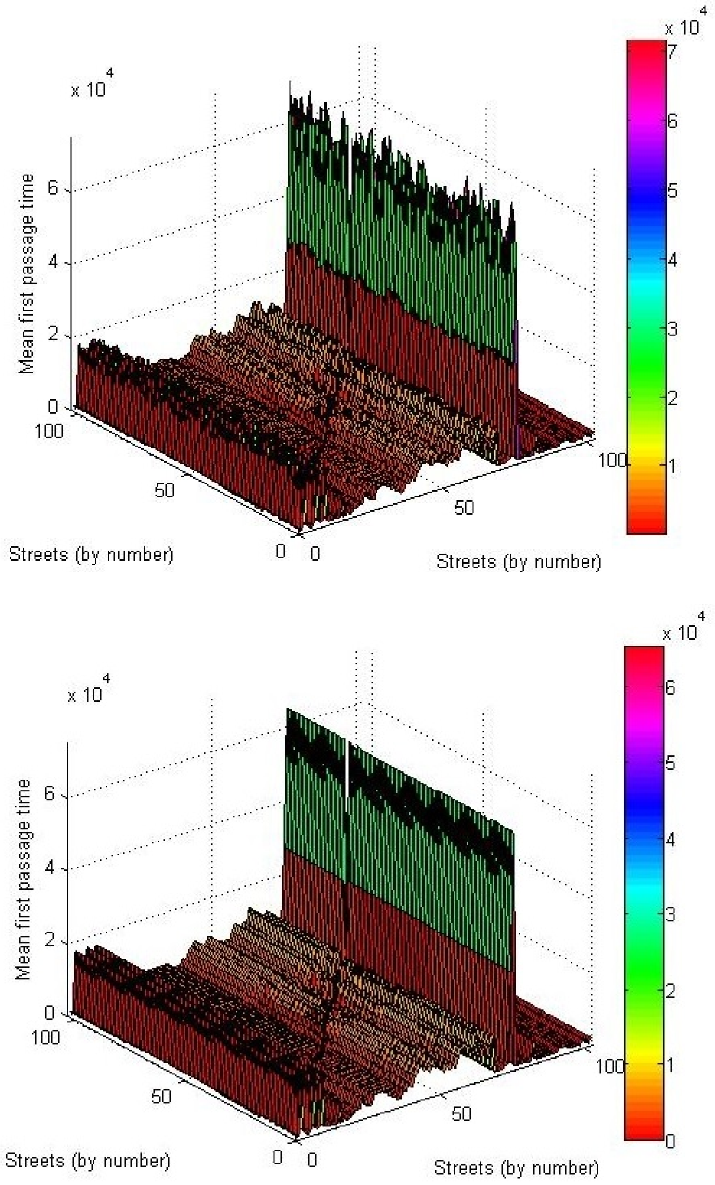
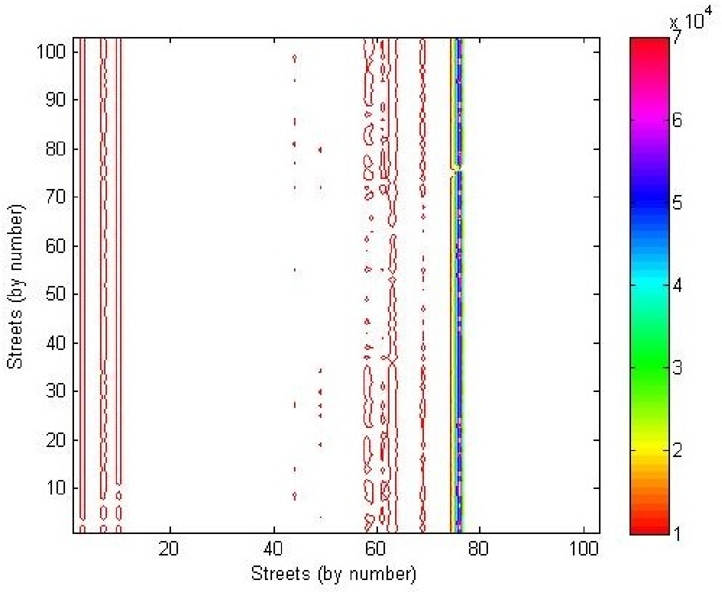


Figure 18: Comparing the surface graphs of the mean first passage time calculated through SUMO (top) and through the Markov chain calculations (bottom).



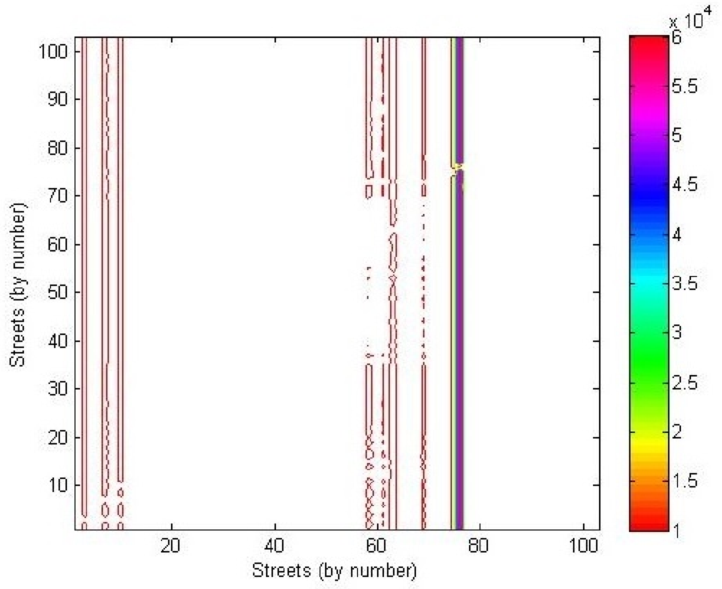


Figure 19: Comparing the contour graphs of the mean first passage time calculated through SUMO (top) and through the Markov chain calculations (bottom).

Figures 18 and 19 compare the mean first passage time matrices found in these ways and once again, the graphs are very much alike. As one can see, the values found through the simulations are a bit more random, shown by the spikes the surface graph has and which are particularly clear when one looks at the higher values. The values found through the Markov chain calculations are much more constant as can be seen by the straighter lines, especially at the higher values. This can be explained by the fact that the simulation is slightly less reliable due to the fact its values are not constant. With every simulation comes slightly different results, sometimes lower, sometimes higher. The Markov chain calculations will always give the same results, which is also seen by the nearly constant outcomes.

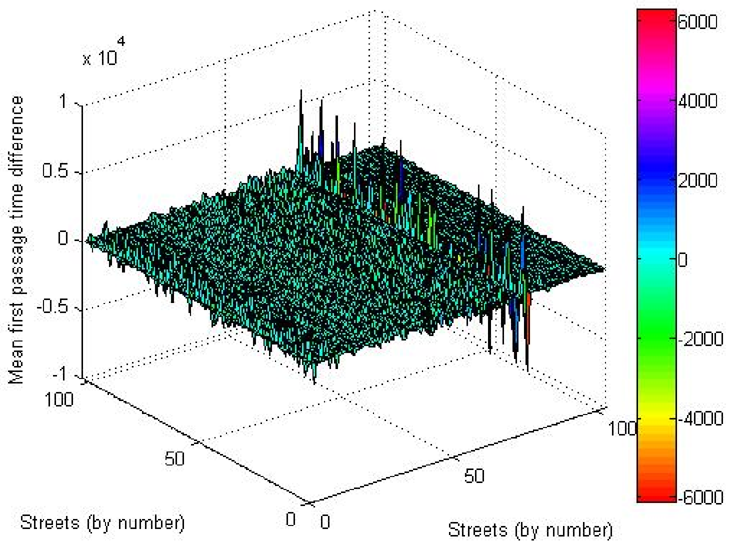


Figure 20: Comparing the difference between the mean first mean passage time matrices.

Figure 20 shows the surface graph of the difference of the values found. As one can see, the same can be said as with the stationary distribution. The values seem to be very close to 0 except for a few spikes. Comparing the different entries of the two matrices, I found that the values found through the mathematical computation differ by no more than 10% from the values found through the SUMO simulations. Although this is less accurate than the values found calculating the stationary distribution, it is still quite decent, although the question arises whether this becomes less accurate when the size or the complexity of the network grows.

3.2.3 Kemeny constant

Since the Kemeny constant is supposed to be, as the name suggests, a constant. It will only be interesting to see if the Kemeny value calculated with the values found through the Markov chain calculation are (close to) constant. The values itself I calculated through the method I explained in paragraph 1.1.3.

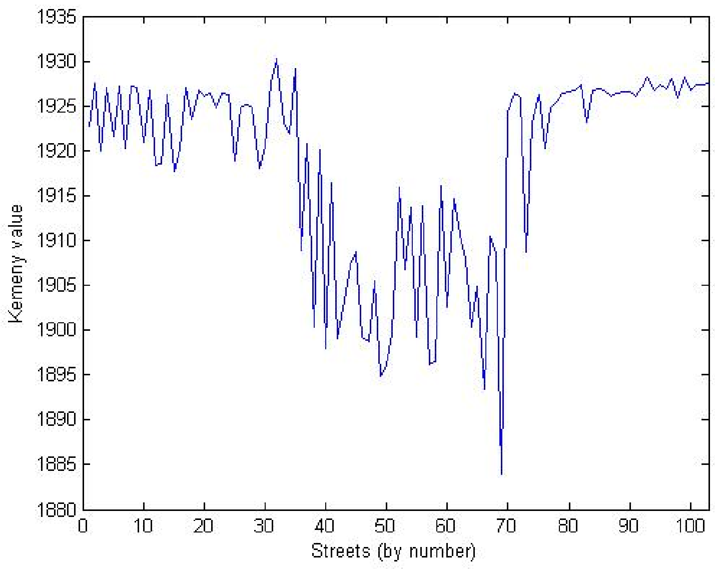


Figure 21: The Kemeny values for the different roads

The values seem to be, except for a few spikes once again, pretty equal. The values are mostly found around 1925, with spikes going as low as 1884 and as high as 1930. This interval of 50 means a difference of around 2.6%, which is once again reasonable accurate. Since the Kemeny constant is calculated using the mean first passage time matrix and the paragraph 3.2.2 showed how these values differed most around the $70th$ street, this accounts for the difference found in these calculations as well.

4 Conclusion and further work

Overall, the results seem reasonably clear and they are pretty accurate when we compare them to the results from the SUMO simulation. The increased size and complexity of the network, by adding lanes, priority regulations etc. will show differences of around 2% when it comes to both the stationary distribution and the Kemeny constant. This is fine, seeing as the original article showed the same differences for these values. For the mean first passage time matrices, we can see the original article has had some differences as well, although it is harder to accurately see how big these differences were. Even so, it is visible how a few values seem to differ more than the 10% I found as a maximum difference in my experiment. So, all in all, it seems that adding all this information into the network does not have too much effect on the accuracy of the calculations.
Taking this into the future, enough possibilities of further work arise. Although this article shows how a bigger network does not have to be a problem, just De Uithof still is not a very big area if you compare it to an entire city. A system like this will only be applied when it can be shown to work on a very large scale. Further work can be aimed at increasing the scale even more, first to entire (large) cities and then to countries and even continents. Next to increasing the size, the complexity can be increased as well. In this article I have added some 'constant' difficulties, like lanes. Another possibility is adding 'variable' difficulties like traffic lights and pedestrian crossings. These obstacles are not constant in a way that they cannot be predicted. Traffic lights change to red when there are no longer any cars waiting to cross the junction or change to green immediately once a car approaches if no other cars are waiting. The same can be said about pedestrian crossings, the arrival of pedestrians at crossings like these are random. They can be approached mathematically, but factors like the weather will also play a part in this: when it is raining heavily, fewer people will be walking outside, meaning fewer people are crossing the road at pedestrian crossings.
In the end, all of this will need more time. Not only to be worked out further, but also for people to get used to the idea. If we look at the way the

authors of the original are planning to apply their ideas, namely by installing a box into every car collecting the needed data, people are not yet ready for this. Due to all the discussion around privacy we have had in the last few years, having a box in your car which will make sure you will be followed everywhere you go will cause even more complaints and discussion. As soon as privacy is no longer an issue like it is right now, applying systems like this might just be a great way to lessen the traffic problem. Until then, there will be no other solution than to sit and wait in traffic, just like the rest of us.

Bibliography

[1] Michael Behrisch, Laura Bieker, Jakob Erdmann, Daniel Krajzewicz; *"SUMO - Simulation of Urban MObility: An Overview"* In: SIMUL 2011, The Third International Conference on Advances in System Simulation, 2011, 63-68.

[2] Ben-Israel, A. (2008). *The group inverse* [Powerpoint slides]. Retrieved from http://benisrael.net/GI-LECTURE-6.pdf

[3] Brin, S., and Page. L. (1998). *The Anatomy of a Lagre-Scale Hypertextual Web Search Engine/* Computer Networks and ISDN Systems 35.

[4] CBS (2012). *Personenautobezit van huishoudens en personen.* Retrieved from http://www.cbs.nl/NR/rdonlyres/69B7DBF3-BA02-4B1F-90D0-40F362C6C4E1/0/2012k1v4p34art.pdf

[5] Cho, G.E., and Meyer, C.D. (2001). *Comparison of Perturbation Bounds for the Stationary Distribution of a Markov Chain.* Linear Algebra and its Applications, 335, 137-150.

[6] Crisostomi, E., Kirkland, S., Shorten, R. (2011). *A Google-like model of road network dynamics and its application to regulation and control.* Internat. J. Control 84 (2011), no. 3, 633651, 90B20 (60J20).

[7] Fiore, M., and Härri, J. (2008), *The Networking Shape of Vehicular Mobility*, in ACM MobiHoc, Hong Kong, China, May 2008.

[8] Google Maps (2010). [De Uithof, Utrecht, The Netherlands] [Street map]. Retrieved from https://maps.google.com/maps?q=De+Uithof &hl=en&ll=52.085954,5.1775&spn=0.011168,0.033023&sll=41.747334,-72.688916&sspn=0.01356,0.033023&hnear=Uithof,+Oost,+Utrecht,+The +Netherlands&t=m&z=16

[9] INRIX (2012). *Files dalen wereldwijd: INRIX Traffic Scorecard geeft een onthullende blik op de worstelende economien in Europa.* Retrieved from http://www.inrix.com/pressrelease.asp?ID=165

[10] Ipsen, I.C.F., and Meyer, C.D. (1994). *Uniform stability of Markov chains.* SIAM J. Matrix Anal. Appl. 15 1061-1074

[11] Kemeny, J.G., and Snell, J.L. (1960). *Finite Markov Chains.* Princeton: Van Nostrand.

[12] Meyer, C.D. (1975). *The Role of Group Generalised Inverse in the Theory of Finite Markov Chains.* SIAM Review, 17, 443-464.

[13] Ross, S. M. (2007). *Introduction to probability models*, 9th edition, Academic Press, San Diego, CA, USA.

[14] Levin, D. A., Peres, Y., Wilmer, E. L. 1., & Peres, Y. (2009). *Markov chains and mixing times:* . Providence, RI: American Mathematical Society.

Appendix

Can be found at:
https://sites.google.com/site/expandingappendix/home