Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Stochastic Simulation of Epidemic Outbreak in a large Network

Jordi Christian, Schmid Yannick & Stücheli Pascal

Zürich
10. Mai 2012

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

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

Author(s)

Last name	First name

Supervising lecturer

Last name	First name

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.

Place and date	Signature

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form

Agreement for free-download

We hereby agree to make our source code of 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.

Jordi Christian

Schmid Yannick

Stücheli Pascal

Table of contents

Abstract

In this study a self-made approach was used to simulate a stochastic epidemic in a scale-free network. The network contained 10,000 nodes representing cities with a total global population of 93 Million people. Firstly, 80 days were simulated in order to quantify the stochastic effect among the simulations. Mean and expectancy range were calculated for the time courses of the ratio of total infected people over global population and for the number of cities with at least ten percent infection. The expectancy range until at least 1,000 people (globally) became infected was $t_{1000\ infected} = 18 \pm 5d$. Secondly, we found that the propagation speed of the disease correlates only weakly with the sum of the degree of the city, where the infection started, and the degrees of its direct neighbours, but is highly affected by stochastic effects. Finally, we found the linear dependency of the time until the appearance of an infected in a random target city and the distance, which has the units of edges, to the city where the epidemic startet to be $t = distance * [6.6 \pm 0.4] + [8.8 \pm 2.4]$.

1 Individual contributions

We initially divided the programming part which was easily feasible since there were three main tasks. Pascal Stücheli was responsible for the epidemic model and its implementation, Christian Jordi familiarized himself with Gephi and was responsible for the network generation. Yannick Schmid wrote the transport function. At any time we assisted each other because we had to make sure that the functions worked well together. Further we had to increase the overall performance of our simulation, mainly by making approximation or changing the structure of certain functions. At this point it was important to discuss the possibilities and limitations of the approximations and all of us contributed ideas and concerns. The analysis and interpretation of the results and also the writing of the report was done in a very cooperative manner.

2 Introduction and Motivations

Last winter, the movie *Contagion*[1] came to the cinemas. The movie is about the fates of some characters during the outbreak of a deadly viral infection disease. The first infected person in the movie was a travelling business woman, who carried the virus across two continents such that the disease could spread over the whole world very rapidly. After seeing this movie we asked ourselves how a realistic disease would behave if no remedy or cure could be found. Inspired by this movie, we thought about a large network representing the big cities on the world, which are either connected by air route or other means of public transportation.

Surveillance and simulation of infectious disease are necessary to predict outbreaks of epidemics and to learn about their dynamic behaviour. At the moment there are of course already several epidemic simulation tools. Some of them also contain stochastic effects like example given *FluTE*[2]. Therefore we do not claim to to reinvent the wheel, but we wanted to see if we would be able to implement an epidemic simulation in a large network with ordinary computational power.

The aim of this study is to follow the time evolution of discrete numbers of people who are either susceptible or infected and the spreading of infected people among a large scale-free network.
We wanted to investigate the following questions:

- What is the impact of the stochastic? How does it influence the dynamics of the epidemic?
- How does the number of total infected in a network vary for a given time of simulation?
- How does the time vary among the different simulation until a certain portion of population in the network has been infected?
- What are the variations for certain cities with different degrees?
- Can we gain predictive power from our result? For example, can we predict the time until a city becomes infected depending on the degree of the first infected city?
- What are the constraints of using a scale-free network?

Additionally we were also interest if we could handle such an amount of data. Actually we spent a lot of time improving the performance of our simulation and exploring approximations. We want to emphasize that we tried to realize our own ideas to create a stochastic epidemic simulation.

[1] *Contagion*. Dir. Steven Soderbergh. Perf. Matt Damon, Kate Winslet, Jude Law. Warner Bros. Pictures, 2011.
[2] Chao DL, Halloran ME, Obenchain VJ, Longini IM Jr "FluTE, a Publicly Available Stochastic Influenza Epidemic Simulation Model." PLoS Comput Biol **6**(1), 2010.

3 Description of the Model

3.1 The continuous disease model

Our model is basically a simplification of the SIR model proposed by Kermack and McKendrick [1]. The major difference is that we did not consider the state R which represents removed people which became immune or died after being infected.
To derive the disease model a city with N inhabitants is considered. The inhabitants are either susceptible for the disease or they already carry the disease and can therefore infect susceptible people. In the equations below susceptible people are labelled S and infected people I. At a constant *birth rate B* healthy (and therefore susceptible) children are born while dying is a first order process with a *death rate d*. The amount of newly infected people per time unit depends on the availability of susceptible people and their meetings with infected people. The *contact rate β* is the rate at which susceptible people become infected after a meeting with an infected person.

$$\frac{dS}{dt} = B - \beta SI - dS$$

$$\frac{dI}{dt} = \beta SI - dI$$

For further simplification we assume that the number of newborn balances the number of dyeing people per time unit in order to neglect the birth and death in the city.

$$\frac{dS}{dt} = -\frac{dI}{dt}$$

$$\frac{dI}{dt} = \beta SI$$

3.2 The stochastic disease model

Since the goal of this study is to follow discrete numbers of infected people and to integrate noise, we obviously ended up with a stochastic model. For a proper stochastic simulation spatial homogeneity is assumed. This means the population in a city is well mixed and there are no spatial accumulations of either susceptible or infected people.
In the SI model there is only one transition possible.

$$s_j + i_k \overset{a}{\rightarrow} i_j + i_k$$

Where s denotes a *susceptible person j or k* and i denotes an *infected person j* or *k*. The *propensity a* is the probability per time unit for a successful infection. The propensity depends on the meeting events between susceptible and infected people as well as on the rate of successful infections after meeting. The most accurate way in calculating dI for only discrete dI and dt would be to calculate in each time step for each infected the number of meetings and then for each meeting the probability whether it leads to an infection or not. With a uniformly distributed random variable then it could be decided whether or not an infection happened through this event. The formula for calculating the probability of a meeting to result in an infection looks as follows (where N is the population in a city).

$$p_I = \frac{S}{N}\beta$$

Each outcome of this decision influences the following decisions. This is an extremely slow calculation for a high population and would not be possible to be calculated. Instead approximations could be used which work with normally, binomial or hyper geometrically distributed random variables. The binomial and hyper geometric distribution can be explained with a series of flipping a coin or drawing balls from which several have the desired colour. In the binomial distribution the different events are independent from the previous ones and in the hyper geometric distribution they are not. The fact that the normal distribution is calculated much faster than the binomial distribution, which is much faster calculated than the hyper geometric distribution, can be used to approximate slow functions with faster ones.

3.3 The network

Instead of modelling an existing city network an artificial scale-free network was created. A scale-free network was chosen, because its capability to form hubs, which we compared with metropoles that are highly interconnected and surrounded by more isolated cities of smaller size. Hence the network would at least be a sufficient model for the air traffic between the cities, as stated by M. Beliant et al. [2].

The scale-free network was generated using an implementation of the Barabási-Albert algorithm [3]. The idea of this algorithm is to create a network around a couple of seed nodes by adding new ones step by step. Thereby the probability to form an edge to target existing node j is given by:

$$p_j = \frac{k_j}{\sum_i k_i}$$

In this way nodes with a high *degree kj* are more likely to be selected and a scale-free network is created. Initially, a loop free version of the algorithm was tested but the created network did not look interconnected enough. Therefore, a loop forming version was used in the simulation (Fig. 1).

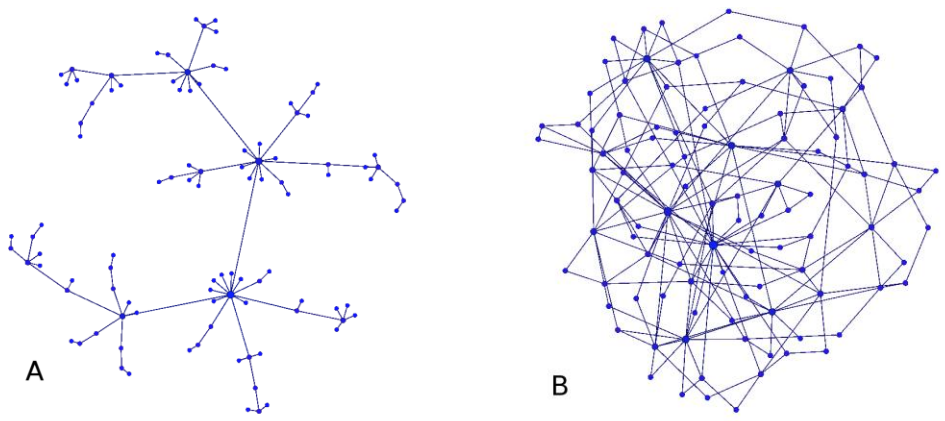

Fig. 1 *Two examples of 100 node networks generated by a Barabási-Albert Model. A: loop free version. B: loop forming version. Gephi 0.81 beta was used for graph visualization.*

It is well known that the population of cities can be described by a power law (Zipf's Law). Just T. and Stephan P. determined the Zipf parameters of german cities with high accuracy [4]. We assigned a population to each node according to its rank in the network and the value predicted by Zipf's Law.
The connection between the population of a german city and its rank is given by:

$$\ln(population) = 15.1 - 0.81 * ln(rank)$$

Our final network consisted out 10 000 cities and had a population of 93 Millions.

3.4 The transport

People from a city x are able to travel to city y if the nodes x and y are connected by an edge. However, we want the total population of a city to be constant. Therefore there the same number of people travel from city x to city y as vice versa while the composition of the travellers (susceptible or infected) can vary. If a fixed number of total travellers per simulation day are assumed – like for example seats in airplanes – the portion of infected travellers is hyper geometrically distributed. T_{xy} corresponds to the *total travellers on the edge between city x and city y* and is equivalent to the number of drawings without laying back out of a population with $S_{x,y}$ + $I_{x,y}$ people and $I_{x,y}$ as the number of infected people[3].

[3] The colon of $S_{x,y}$ and $I_{x,y}$ means „or".

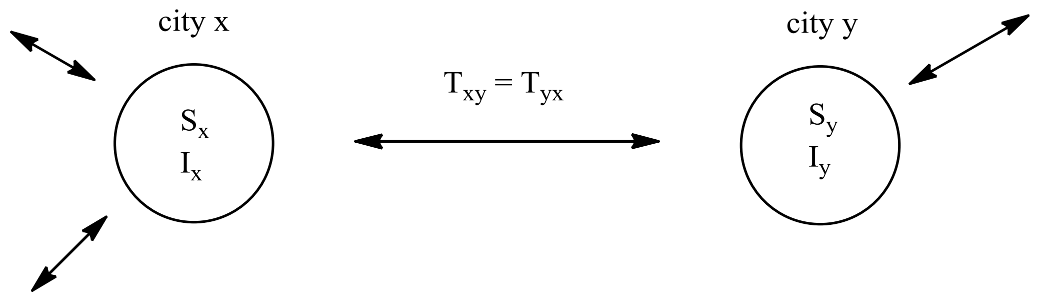

Fig. 2 *Sketch of two cities x and y where S and I correspond to the number of susceptible or infected people. Edges are indicated as arrows. T denotes the total number of travellers along an edge and is the same in both directions of the edge.*

The number of total travellers along an edge should obviously not exceed the population of either city connected by the specific edge. It must also be ensured that cities with multiple edges do not have higher traveller fluxes than populations. Also a factor depending on the *degrees k* of the connected cities compared to the average degree in the network was included to contribute more transports between cities with many connections – intuitively that is like cities with more airports and stations have more transport capacity. Finally we included a factor to reduce the transport of infected people assuming that people who feel sick travel less.

$$T_{xy} = \frac{k_x + k_y}{< k >} * N_{x,y} * \frac{1}{k_{x,y}} * \frac{dt}{24}$$

Actually, it would have been nicer if the degree factor would be divided by two. However, since the transport rate is further reduced[4], this flaw is corrected.

[4]The factor is not shown in the equation above, but was included in the simulation (see section A.6, FINAL_SIMULATION_whole_run.m).

4 Implementation

4.1 Network generation

The function *networkepid2loops* is used to generate the scale-free network. After the definition of the seed nodes and a seed edge list, a network of 10‘000 nodes is created, by introducing new nodes one after another. Each new node is enabled to form two edges to existing nodes. Hence loops can be formed. The targeted loops are chosen randomly and their p_j is calculated. Thereafter, a random number between zero and one is generated and compared with p_j. If the random number is smaller the edge is formed, otherwise a new target is chosen.

The edge list is saved as an nx2 matrix. Every row contains the two nodes connected by the edge. To import the matrix into the graph visualization tool Gephi, it had to be converted using the function *cell2csv* [5].

The nodes are stored in *cities.txt*, an mx3 matrix, where every row stands for a city in the network. The degrees of the nodes stand in the first column of this matrix and are then passed to the function *city_size*

city_size ranks the cities according to their degree and used Zipf’s law to assign a population size to each city. Thereby a variation of 1% is added to the predicted size to reduce the impact of the sequence on cities that have the same degree.

The whole process of generating the network is collected in the function *initialization.*

4.2 Simulation of the infection

As the name indicates, the function *Simulate_Infection* computes the number of infections per city and per time step. Therefore the number of meetings between infected and healthy people is calculated according to an arbitrarily chosen mean and standard deviation of daily meetings with other people. Further, a Gaussian distributed error is included. As long as the number of infected people in a city is below a certain threshold the infection is simulated in a stochastic manner. The number of meetings which result in an infection is a hyper geometrically distributed number, as mentioned in section 3.1. Because MATLAB is very slow in drawing hyper geometrically distributed random numbers, a binomial distributed random number is used as an approximation. The drawback of the binomial distribution in this situation is that a person can be infected more than one time. However, by using small time steps, the probability to infect the same person two times becomes rather low. As soon as the number of infected people exceeds the threshold, the simulation works in a deterministic manner and is equal to the expected value.

4.3 Transport

According to section 3.4 the total number of travellers along every edge is calculated before the actual simulation. Then for every city and time step the number of infected travellers is simulated. The transport function uses the edges matrix and in order to avoid constraints from calculating always in the same order along the edges matrix, the rows of the matrix are shuffled in every time step. Again, we have hyper geometrical distribution for the number of infected travellers. So, the mean and the standard deviation of the hyper geometrical distribution are computed. This is done with the MATLAB function *hygestat*. A Gaussian distributed error is included to introduce noise in the transport. Finally, the number of infected people in the cities is updated due the change through transport.

4.4 Simulation

The function *FINAL_SIMULATION* and its descendants bring the individual parts together. Many different versions of this function were used to perform our experiments. They all follow the same basic structure:

- The network data is imported and the starting point of the infection is generated.
- For each timestep:
    - The transport of infected people takes place (*transport_with_fixed_tot_T*).
    - New inhabitants are infected (*Simulate_Infection).*
    - The new infected population and other data is stored.
- At the end of the simulation the data of interest is exported

Thereby the biggest differences can be found in the last step that is adapted to the goal of the experiment

4.5 Experiments

Three simulations were performed to characterize the epidemic outbreak in the scale-free network. To start each simulation, the first infected person was set randomly into a city – the *seed*. The parameters like infection probability and mean of daily meetings were chosen empirically and tested in smaller networks. Implementation of real parameter values would be interesting for further experiments and is discussed in the section 6.

In the first experiment 80 days were simulated and the number of cities with at least ten percent infected citizens was taken as output. Additionally, the ratio of total infected people over the whole network population was computed. The aim was to compare the time courses of the epidemic in multiple simulations and to measure the temporal variations.

The second experiment was performed to test the hypothesis that the epidemic spreads faster when started big cities with high degrees than in small cities. Since, we assumed a high influence of the nodes connected to the seed, the sum of the degrees of the direct neighbours was considered too. In every simulation, the time until at least twenty cities had at least one infected person was measured.

In the third experiment, in addition to the seed, a random target city was chosen. In order to find the number of edges in the shortest path connecting the seed with the target city, which is the distance, we used the MATLAB Graph theory tool box. The function *graphshortestpath* solves the shortest path problem. Then, the time until at least one infected person appeared in the target city was measured.

4.6 Movies

To get a nice view over the performance of the simulation, a short movie of the progressing infection was generated. Therefore a small network with 500 nodes was created and simulated once over a period of 80 days. Using the function *output2gexf* the generated data was converted into a dynamic gephi network. Subsequently, Gephi was used to present the course of the infection. The percentage of infected citizens is indicated by the city colour, reaching from blue (not infected) over green and yellow to red (heavily infected). The scale is not linear; the change in colour is faster for low infected numbers to make the time point when a city gets invaded clearer. The Gephi animation was recorded using 'Debut Video Capture Software'.

The video can be found in our git hub repository

5 Simulation Results and Discussion

5.1 Simulating 80 days

The aim of this experiment was to follow the time course of the whole epidemic over a given time of 80 days and to see the impact of the stochastic effect among the different simulations. We were interested, on one hand in the regional progression of the disease, which we tried to show with the number of cities with at least ten percent infected citizens. On the other hand, in the temporal variations of the number of global infected people. For both, the ratio of global infected people and the number of cities with at least ten percent infected people the mean was calculated (Fig. 3 and Fig. 4 respectively). To show variations in time, either for a certain ratio or a number of cities with at least ten percent infection, the temporal expectation range was computed (Fig. 3 and Fig. 4 respectively). Further, the regional and the quantitative progress of the disease were compared (Fig. 5). To gain a quantitative value for the stochastic, the times until 1000 people in the network became infected were used to compute an expectation range, $t_{1000\,infected} = 18 \pm 5d$.

The time courses of all simulations (Fig. 3 shows an extract) looked very similar in terms of slope and shape. Apparently, the main difference was the start of the "exponential" progression. As soon as the disease switches to the fast progression, the stochastic loses its impact. Obviously, we assume some influence from the degrees of the seed city and its neighbouring nodes (see section 5.2), because for every simulation a random seed was used. Secondly, the fluctuations must play a major. Until around the 35th day, (Fig. 5), a susceptible person rarely meets an infected. During the slow progression phase of the disease, single events like a meeting, which resulted in an infection or an infected person, who successfully travelled to another city, greatly influence the later outcome of the epidemic. The expectation range of ten days for the time until 1000 people are infected supports this suggestion. With the computed expectancy ranges, although they are relatively broad, the transition to the fast progression of the disease can be predicted.

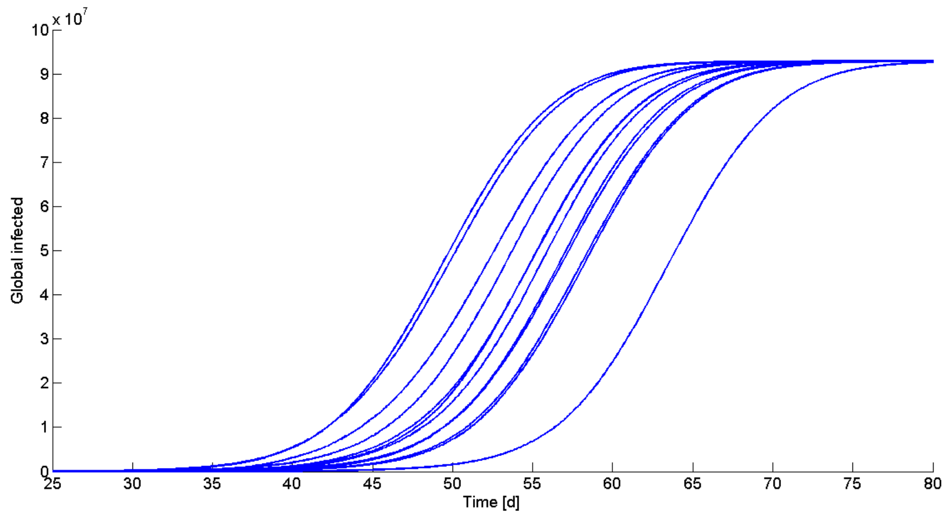

Fig. 3 *Some examples of time courses from simulating the epidemic for 80 days. The curves have similar shapes. The major difference is the time of the transition to a rapidly spreading epidemic.*

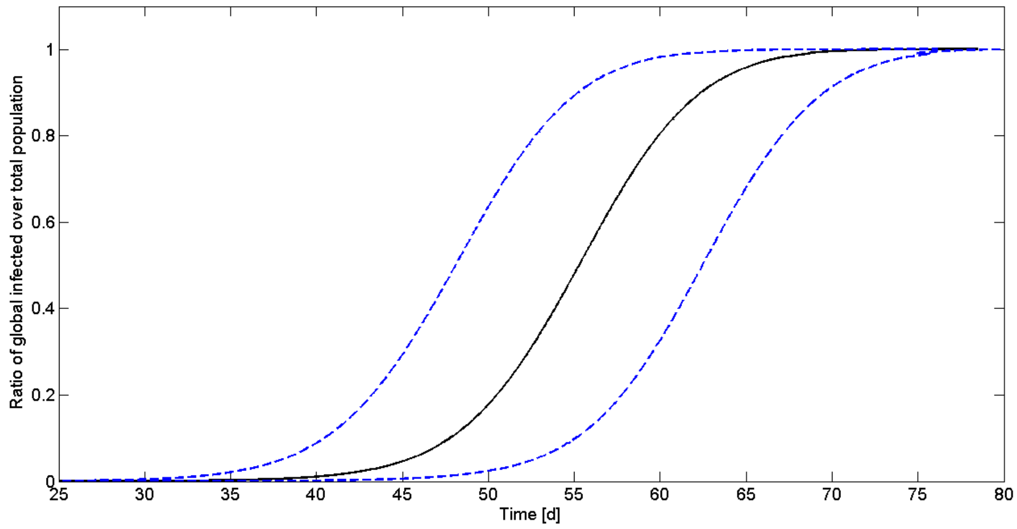

Fig. 4 *The mean of the time course of the disease for 80 days is shown as black line by plotting the average ratio of the total number of infected people vs. the time. Further, the expectation range of the time at which a certain ratio is reached, is shown as dashed blue lines.*

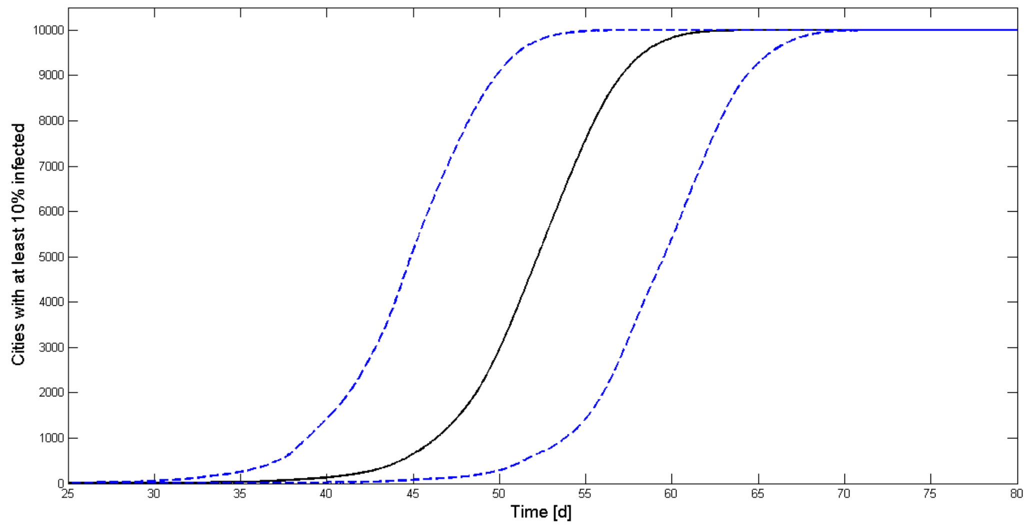

Fig. 5 *This figure provides the mean, black line, and the expectation range, dashed blue lines, of the time, which is needed to infect at least ten percent of the citizens of a certain number of cities.*

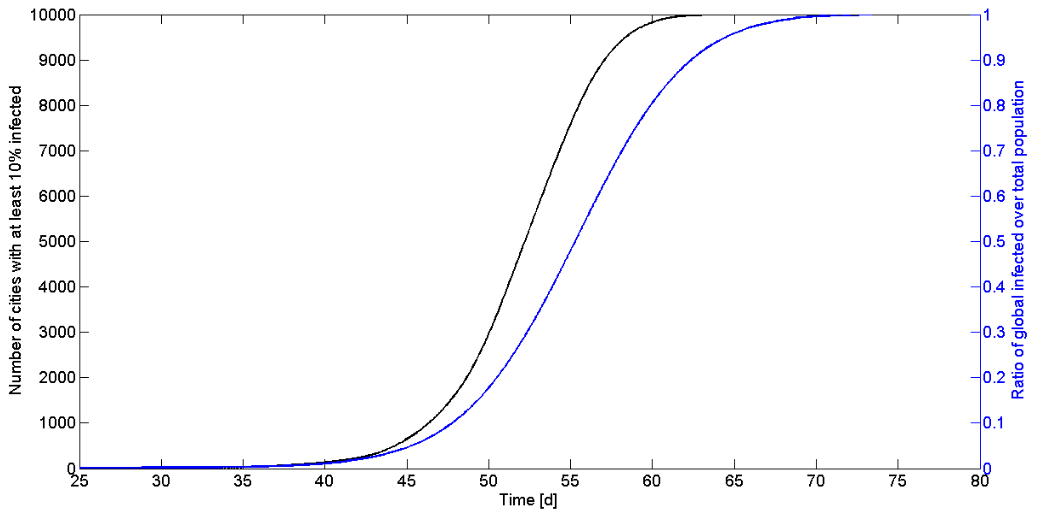

Fig. 6 *Simulations of the epidemic for 80 days. The mean of the number of cities with at least ten percent infected citizens after simulating 80 days is shown by the black line. In addition, the mean of the ratio of the total number of infected people over the total network population is represented by the blue line. The means were calculated from the data of 147 simulations.*

5.2 Degree correlation

The aim of this experiment was to correlate the total degree of the city, where the seed was set, and its direct neighbours with the speed of the disease spreading. The time needed to infect at least 20 cities with at least one infected person was measured and plotted against the total degree (Fig. 6).

After performing several regressions on the data two things became obvious. The spread of the data dominates but still a certain tendency is observed. The power law regression with an R^2 of 0.36 fits best but nevertheless not fully convincing, since we expected a higher influence from the total degree. In low degree local networks one expects a slow propagation of the disease, because in cities with a low degree, transport of infected people is less probable. Further, the possible directions are also limited (Fig. 7, left) compared to a city with a high degree (Fig. 7, right). Whereas the infected people in cities with high degrees have a higher probability to travel and more edges to travel along.

A reason for the low correlation may be that the experimental setup was not detailed enough. Maybe, the 2nd generation neighbours also need to be taken into account for further experiments. As a conclusion of this experiment it can be said that the starting point of the epidemic influences the disease spreading minorly. In other words, from only observing the very local surrounding the dynamics of the epidemic cannot be foreseen well. In the used network, every node is connected in average over five other nodes with a random chosen node. This reflect the high connectivity in the real world and contributes to the fact, that as soon as a certain number of infected is reached, the disease spreads rapidly in the network.

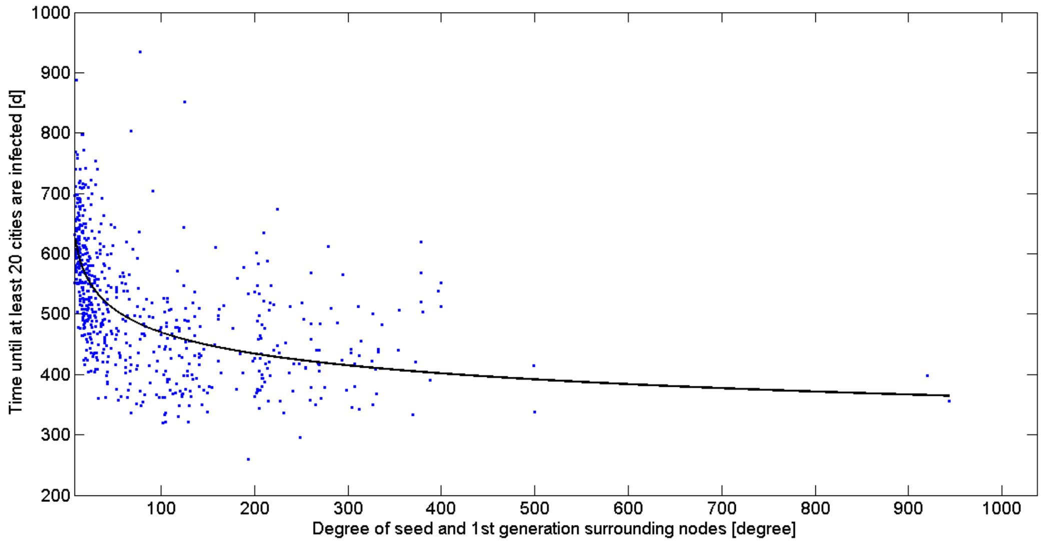

Fig. 7 *The time until at least 20 cities having at least 1 infected person vs. the sum of the degree of the seed and its first generation neighbors is shown in blue for 635 simulations. The black line represents the curve fit performed with MATLAB, assuming a power law dependency. Due the lack of measurement points between degrees of 400 to 900, the statistical power of the fit might be limited. Nevertheless, a certain tendency is observable.*

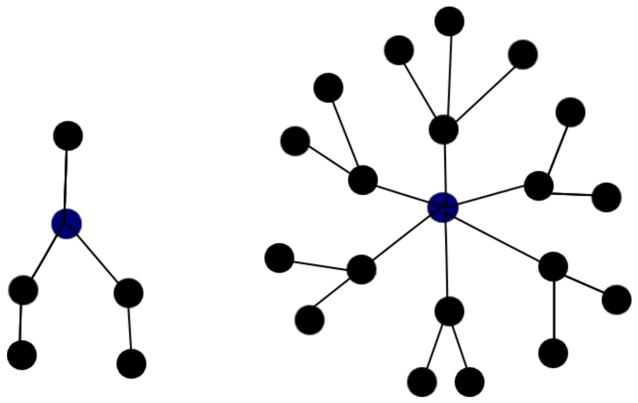
Fig. 8 *An infected city shown in blue and its neighborhood. Left: A neighborhood a low total degree. Right: A neighborhood with a high total degree. For the degree correlation experiment the sum of the degree of the blue city and the degrees of the 1st generation neighbours was assigned as total degree and correlated with the spreading speed of the epidemic.*

5.3 Infection over distance

The intention of this experiment was to test wether the network or the transport function generated unintentional constraints. We intuitively assumed that the time to infect the random chosen target city increases with its distance to the seed. Since, we used unweighted edges the distance is equal to the sum of edges between seed and target city, where every edge counts as a one.

From the results of the simulations the mean and standard deviation were computed (Fig. 8). The axial intercept scatters relatively high compared the the slope. This is due the standard deviation, which has about the same range for every distance. To quantify the tendency a linear regression analysis was performed, which revealed the linear dependency $t = distance * [6.6 \pm 0.4] + [8.8 \pm 2.4]$. This formula allows the prediction of the arrival of the first infected person in a target city in the network.

The average distance of the used network was five and the diameter was nine. There was no data for distance equal one, because the probability to choose a target city which is a direct neighbour of the seed is very small in a network with 10,000 nodes. Same counts for distances above seven.

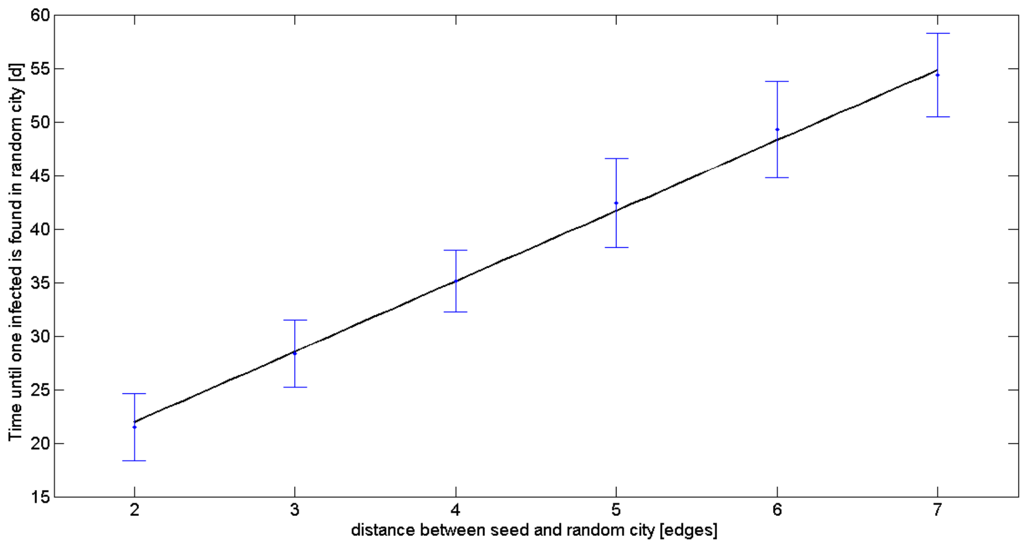

Fig. 9 *This figure provides the correlation of the distance and the time needed for the disease to reach the random target city. The distance is the shortest path from the seed (random city where the first infected person was set) to a random target city and corresponds exactly to the number of edges, since we used un-weighted edges in our network. The error bars and the mean are shown in blue. The black line is the result of the linear regression analysis of the time measurements city ($t = distance * [6.6 \pm 0.4] + [8.8 \pm 2.4]$).*

6 Summary and Outlook

6.1 Summary

Our model showed big fluctuations in the speed of the disease spreading (Fig. 3, 4 and 4). The broad expectations range shows that the average infection course is an inaccurate indicator for the outcome of a disease outbreak. These differences in the course of the infection could have two reasons:

One explanation could be the randomness of the model. Especially at the beginning of the simulation low infected numbers can lead to big variety. This fact was measured either graphically with Fig. 3+4 or with one «slice» through the plot. For example the expectancy range of the time until 1000 people are infected is $t_{1000\,infected} = 18 \pm 5d$. This effect decreases in later stages as higher numbers of infected people are reached. This is because it will average itself out and then the approximations in our model will take over. This leads to the disease outbreak dynamics always looking the same above a certain threshold. The only thing that varies is the time when this continuos deterministic behaviour starts (Fig. 3).

Another factor is the initial point of the infection. The experiments show the tendency that the speed of the outbreak depends on the connectivity of the starting city (Fig. 7). The higher the connectivity is the faster is the disease spreading. It is clear that in order to confirm these dependency further experiments have to be done. In the last experiment we found that the disease progresses linearly around the root of the disease (Fig.9). The further away a city is from the root (in units of edges which have to be passed), the later the disease will arrive there.

The created movie makes it possible to track the disease spreading at the beginning of the simulation. It can be seen that the simulation produces plausible and comprehensible results. Also the experiments confirmed expectations. Although this simulation is only a primitive model of reality, some of its results might also applicable to the real world. For example it makes intuitively sense that diseases that arise in traffic junctions will spread faster trough the world. Also it is shown that a purely deterministic prediction of disease outbreaks is not enough.

6.2 Outlook

In our experiment a completely fictional network was used, so obviously, the next step would be to implement real data and an existing network. For the network it would make sense to take a continent or even the whole world as long as only the important cities are considered, so that the performance does not suffer too much. It would be interesting to weight the edges according to geographic distances and to include real transportation data. Furthermore, the integration of either a healing rate or an immunisation would extend the simulation and make it more realistic.

We assume that infection simulation will gain further importance for the understanding of the spreading in network. Either one can think of diseases like H1N1 or sexually transmitted diseases like HIV. But also the spreading of information or malware in computer networks could be simulated with programs, such as ours.

7 References

[1] Kermack WO and McKendrick AG, "A Contribution to the Mathematical Theory of Epidemics." Proc. Roy. Soc. Lond. A **115**, 700-721, 1927.

[2] Berliant M. and Watanabe H, "A Scale-Free Network Structure Explains the City-Size Distribution." mimeo **1-11,** 2008.

[3] http://www.bsse.ethz.ch/cobi/education/Math_Mod_basic/ex8_2011.pdf (25.5.2012, 0100)

[4] Just T and Stephan P, "Die seltsam stabile Größenstruktur deutscher Städte: Das Zipfsche Gesetz und seine Implikationen für urbane Regionen." Deutsche Bank Research, No **31**, 2009; http://www.dbresearch.de/PROD/DBR_INTERNET_DE-PROD/PROD0000000000242036.PDF (25.5.2012, 0100)

[5] https://github.com/msssm/lectures_files/blob/master/networks/cell2csv.m (25.5.2012, 0100)

[6] http://www.soms.ethz.ch/teaching/MatlabSpring12 (25.5.2012, 0100)

[7] Balcan D, Hu H, Goncalves B, Bajardi P and Poletto C, "Seasonal transmission potential and activity peaks of the new influenza A(H1N1): a Monte Carlo likelihood analysis based on human mobility." BMC, 2009.

A MATLAB code

A.1 Initialisation

```matlab
function[]=Initialisation()

%Summary of the functions needed to create our start network. bcs
%everything needed is stored in .txt or .csv files. -> has only to be
%executed once

%This function generates:   -cities.txt w/o infection
%                           -edges.txt
%                           -network.csv
%                           -tot_T.txt

ncit = 10000; %number of cities in the Network
seeded = [1 5; 2 3; 4 5;4 3];%seed for the edges
seedct = [1 0 0;1 0 0;1 0 0;1 0 0;2 0 0]; %seed for the cities
dt=2; %2hours (timesteps of the simulation) -> needed to scale the
      %passenger volumina

%Generation of edge and node lists of a network with loops
[cities,edge]=networkepid2loops(ncit,seeded,seedct);

%Generation of the city populations
cities=city_size(cities); %output is cities.txt

%Generation of the traffic volumina on each edge
tot_T = generate_fixed_tot_T(cities,edge,dt);

%Stores the traffic volumina for the simulation. The other data sets
%were already stored by the functions generating them.
dlmwrite('tot_T.txt',tot_T);

end
```

A.2 networkepid2loops

```matlab
function[cities,edge]=networkepid2loops(ncit,seeded,seedct)
%Generation of a scale-free network, with the possibility to form loops

%Input parameters are:
%ncit: integer, number of nodes in the generated network
%seeded: seed for the edges matrix. array of size (nx2)
%seedct: seed for the city matrix. array of size (mx3)

%Output is:
%cities: matrix of size ncitx3, the first column is filled, the two
%others are blank
%edge: matrix of size yx2, undirected edge list of the graph
{city,city2;..]
```

```
%Additional output: -a .txt file of the edge list to be used by other
%                       functions
%                   -a .csv file of the edge list, to be used by gephi

cities=zeros(ncit,3);
pos= length(seedct(:,1));
cities(1:pos,1:3)=seedct; %Stores (degree,Inhabitants,Infected)
edge=seeded; %Connections btwn the cities, store lower index first
pass=0; %number of passengers on an edge %%%%%not used yet%%%%%%%%
mlinks=2; %number of links that are fixed at each step increased to two
sumlinks=length(edge(:,1)); %total number of edges
linkage=0; %Determines wether a new edge has been set

%functions
while pos < ncit
    pos= pos+1;
    posed=length(edge)+1;
    linkage=0;

    while linkage ~= mlinks %true as long as the new edge has not been
                            %set

        %pos= pos+1;
        %posed=length(edge)+1;

        rnode = ceil(rand * pos); %randomly choose a node in front of
                                  %the current position
        deg = cities(rnode,1); %degree of the chosen node
        rlink = rand; %random number to determine wether an edge is
                      %formed

         if rlink < deg / sumlinks &&
isequal(edge(length(edge(:,1)),:),[rnode,pos])~=1
            cities(pos,1)=cities(pos,1)+1; %degree of the new
                                           %introduced city is
                                           %increased by one
            cities(rnode,1)=cities(rnode,1)+1; %The deg of the chosen
                                               %city is increased by one
            edge(posed,1)=rnode; %stores the new edge
            edge(posed,2)=pos;
            sumlinks=length(edge(:,1));
            linkage=linkage+1;
            posed=posed+1;
         end
    end
end
%bring data in a form gephi can understand -> convert into a cell to
%use the provided function cell2csv.
edgecell=num2cell(edge);
edgecell2{length(edgecell),3}=0;
```

```
%Beside the Source and Target nodes an additional column is needed so
%that gephi generates an undirected graph
for i=1:length(edgecell)
    edgecell2{i+1,1}=edgecell{i,1};
    edgecell2{i+1,2}=edgecell{i,2};
    edgecell2{i+1,3}='undirected';
end

%The columns need titles, so that gephi recognizes them
edgecell2{1,1}='source';
edgecell2{1,2}='target';
edgecell2{1,3}='type';

%Save the data as a .csv for gephi and as .txt for the simulation
cell2csv('smallernetwork10kfinalloop.csv', edgecell2, [], 2007, [])
dlmwrite('edges.txt',edge);

end
```

A.3 city_size

```
function[cities]=city_size(cities)
%Code to determine the size of the cities

%Input parameters:
%cities: city matrix of format mx3, where the first column is filled
%with the degree of the city in the Network

%Output:
%cities: modified version of the input matrix. The city population
%(cities(:,2)) has been added

%Additional output: cities.txt stores the city matrix, so it can be
%reused.

%rank the cities depending on degree
rank=zeros(length(cities(:,1)),1);
rank(1)=1;
for i=2:length(cities(:,1))
    y=1;
    while y<i
        if cities(i,1)>=cities(rank(y),1)
            for k=(i-1):-1:y
                rank(k+1)=rank(k);
            end
            rank(y)=i;
            y=i; %to break while loop
        elseif y==i-1
            rank(i)=i;
            y=i;
        else
            y=y+1;
        end
    end
end
```

```
%Now use zipfs law to generate the city sizes.
%the parameters ln(a)= 18.6 b=1.23 are used (see 3.3 for further info)

a=exp(18.6);
b=1.23;

for i=1:length(cities(:,1))
    zipf=exp(-log(i/a)/b);
    cities(rank(i),2)= round(zipf+randn*zipf*0.01); %Added 1% normal
distributed noise
end

%Store modified file
dlmwrite('cities.txt',cities);

end
```

A.4 generate_fixed_tot_T.m

```
function tot_T = generate_fixed_tot_T(cities,edges,dt)
%% Function which calculates the number of travellers along every edge.

k = cities(:,1); % degree

N = cities(:,2); % Total population
I = cities(:,3); % Infected

% Calculate the mean of degree in the network
k_mean = mean(k);

% Timefactor
tf = dt/24;

for i = 1:length(edges)

  %For better overview define x as city 1 in the edges matrix and y as
  %city 2 in the edges matrix
    x = edges(i,1);
    y = edges(i,2);

  %We weight the transport along an edge with the degrees of the cities
  %This is due our assumption that the plublic transport to cities with
  %many connection (e.g. big station or airport) must be high.
    trsp_xy = (k(x)+k(y))/k_mean * tf; % factor scaled with degrees.

  %We take the population of the smaller cities to determine the total
  %number of voyagers to avoid that a small city has more leaving
  %voyagers than inhabitants. This is based on our assumption that the
  %number of voyagers in both direction of an edge must be the same to
  %keep the population in the cities constant). Scaling with the
  %reciprocal of the degree of the smaller city ensures that also small
  %cities with high degree do not higher fluxes than citizens.
```

```
    if  N(x) < N(y)
        tot_T(i) = round(trsp_xy * 1/k(x) * N(x));

    else % would be N(y) < N(x) or N(x) = N(y)
        tot_T(i) = round(trsp_xy * 1/k(y) * N(y));

    end

end

end
```

A.5 output2gefx

```
%This function transfers the data, generated by our simulation into a
%.gexf file, that encodes for a dynamic Network. This gexf file can be
%visualized using Gephi.

%The function needs a txt file containing the infected levels in the
%Network at various timesteps, a gexf file containg the network data
%and optional a file containing the total population of all the cities

function[]=output2gexf()

infected=dlmread('exp61new.txt'); %Problem!!! contains NaN...
gexf=importdata('dyndemo500.gexf');
cities=dlmread('cities.txt');
pop=cities(:,2);

cities=length(infected(:,1)); %number of cities in the array
timesteps=length(infected(1,:)); %number of timesteps

%Normalize the infected over the total city population
for i=1:cities
    infected(i,:)=infected(i,:)/pop(i);
end

h=1;
q=1;

%Reduces the total number of timesteps used for the generation of the
%dynamic graph.
while q<=timesteps
    infected2(:,h)=infected(:,q);
    %After q=800 the infection is almost complete -> take less
datapoints
    if q>800
        q=q+50;
    else q=q+1;
    end

    h=h+1;
end
infected=infected2;
timesteps=length(infected(1,:))
```

```
%strings needed for search and the insertion of the data

target='      <node id="1" start="1.0">';
data='          <attvalue for="Infected" value="000000000000"
start="0000.0" endopen="0000.0"></attvalue>';
findata='          <attvalue for="Infected" value="000000000000"
start="0000.0"></attvalue>';

%lgexf2=length(gexf)+cities*timesteps;
gexf2=cell(10000,1);

pos=1;
pos2=1;

%Copies the header of the template gexf file into the new version until
%the data of the first node can be inserted

while strcmp(target,gexf{pos}) ~= 1
    gexf2{pos2}=gexf{pos};
    pos=pos+1;    %15
    pos2=pos2+1; %15
end

gexf2{pos2}=target;
gexf2{pos2+1}='        <attvalues>';
pos2=pos2+1; %Update to the correct position 16

%Insertion of the infected levels at every timestep for each node.

for i=1:cities

    for n=1:timesteps-1
        data2=data;
        dummy=num2str(infected(i,n));
        d=length(dummy);
        data2(55-d:54)=dummy(1:d);
        dummy=num2str(n);
        d=length(dummy);
        data2(68-d:67)=dummy(1:d);
        dummy=num2str(n+1);
        d=length(dummy);
        data2(85-d:84)=dummy(1:d);
        gexf2{pos2+n}=data2;
    end

    pos2=pos2+timesteps; %19

    %The data of the last timepoint needs another format, because it
    %has no endpoint
    findata2=findata;
    dummy=num2str(infected(i,timesteps));
    d=length(dummy);
    findata2(55-d:54)=dummy(1:d);
    dummy=num2str(timesteps);
    d=length(dummy);
    findata2(68-d:67)=dummy(1:d);

    gexf2{pos2}=findata2; %the infected for the first node are now
                          %calculated

    pos=pos+3;%18 %24

    %Copies the positions of the nodes from the template into the new
    %gexf

    for t=1:7
        gexf2{pos2+t}=gexf{pos+t}; %
    end

    pos2=pos2+7;%40
    pos=pos+6;%23

end

%Copies the rest of the template gexf

for y=0:length(gexf)-pos
    gexf2{pos2+y-1}=gexf{pos+y};
end

%The collected data is now exported into a new gexf file

file_1 = fopen('500dynamic.gexf','w');

for u=1:length(gexf2)
    fprintf(file_1,gexf2{u});
    fprintf(file_1,'\n');
end

fclose(file_1)

end
```

A.6 FINAL_SIMULATION_whole_run

```
%This program simulates an epidemic disease outbreak in a imaginary
%environment. With a second program a city network was created with
%traffic between them. The simulation in this program calculates the
traffic and infections with a fast approximated stochastic algorithm.

%Different versions were used % --- indicate another version. If any of
%these functions are used it is not ensured that they work in the
%present version because many small changes allways had to be done

function FINAL_SIMULATION_whole_run
%%
%Each run with b is a whole new simulation for the entire simulation
%duration. Parfor uses all processors the computer provides. To get
%this calculation power bonus the cores must be connected with
%"matlabpool open" Doing this will give a linear speed bonus with the
%number of processors.
parfor b=1:500

    %Load city information
    cities = dlmread('cities.txt');

    %Set random infection root for this experiment
    root= unidrnd(length(cities(:,1)));
    cities(root,3)=1;

    %Load connections between the cities
    edges = dlmread('edges.txt');

    %Load the number of transports per tick per connection between two
    %cities
    tot_T = ceil(dlmread('tot_T.txt')/100);

    %An other version for the distance correlation used a slightely
    %different code in addition. Its just here for a complete code.
    % ---
    %     target_city = root;
    %     while target_city == root
    %         target_city = unidrnd(length(cities(:,1)));
    %     end
    %
    %     testsparse = sparse(10000,10000);
    %     for i = 1:length(edges)
    %         testsparse(edges(i,1),edges(i,2))=1;
    %         testsparse(edges(i,2),edges(i,1))=1;
    %     end
    %
    %     distance = graphshortestpath(testsparse,root,target_city);
    %     testsparse = 0;
    % ---
    %Parameter definition
    dt = 2; %Tick duration hours
    runtime = 24*80; %Whole runtime hours
    t = 0; %Initialization
    meeting_events_mean = 7.5;%Number of meeting events per day mean
    meeting_events_stdev = 7;%Number of meeting events per day stdev
    infection_prob = 0.05; %Infection probability on meeting event
    g = 1; %Initialization
    output_array = zeros(length(cities(:,1)),runtime/dt);
%Initialization
    tot_pop=sum(cities(:,2)); %Calculate total population

    %Go through all timesteps
    while t < runtime
        %% Main step for each time tick

        %1. Step Traffic
        cities = transport_with_fixed_tot_T(cities,edges,tot_T);

        %2. Step Infection
        Cities = Simulate_Infection (cities,dt,gmeeting_events_mean,
        meeting_events_stdev,infection_prob,t);

        %3. Update time and variables
        output_array(:,g) = cities(:,3);

     %For the distance correlation a break condition was used because
     %not the whole simulation had to be run. The break condition comes
     %into play when the target city is infected.
        % ---
        % if cities(target_city,3) > 0
        %   generate_output(output_array,tot_pop,cities,
        %   root,edges,b,t,distance)
        %   break
        %         end
        % ---

     %For the degree correlation another break condition was used
     %because not the whole simulation had to be run. The break
     %condition comes into play when at least 20 cities are infected.
        % ---
        % if length(find(cities(:,3)>=1))>=20
        %   generate_output(output_array,tot_pop,cities,root,edges,b,t)
        %   break
        % end
        % ---

        t = t + dt;
        g = g + 1;

    end

    %Generate what needs to be saved (Whole experiment would be over 50
    %mb for each run, which is not possible to save several 100 times.)
    generate_output(output_array, tot_pop,cities,root,edges,b);

end
```

```
%To enable it to shut down the computer after whole simulation use:
% dos('shutdown /s')
% quit
end
```

A.7 Simulate_Infection

```
function cities=
Simulate_Infection(cities,dt,meeting_events_mean,meeting_events_stdev,i
nfection_prob,t)
%Function which calculates the infection spreading in one city per tick

%Calculate parameters for one tick and initialize
meetings_stdev = meeting_events_stdev/24*dt;
%66 percent of meetings %per day are in mean +- stdev range

meetings_mean = meeting_events_mean/24*dt;
number_of_cities = length(cities(:,1));

%Run through each city
for n = 1:number_of_cities

    %% Calculate infections for each city

    %Set temporary parameters
    infected = cities(n,3);
    susceptible = (cities(n,2)-infected);

    %Don't go on if there are no infected or no susceptibles left in
    %the city (performance)
    if infected > 0 && susceptible > 0
       %%
       %If infected are below 10000 in the city calculate with a random
       %part in the calculation
        if infected < 10000
        %%
        %Calculate meeting events between infected and any other people
        %Normally distributed, rounded and abs taken
            meeting_events = round(abs(randn*meetings_stdev*infected +
                                  meetings_mean*infected));

            %Calculate meeting events which result in an infection
            %(approximation because detailed calculation would not be
            %possible for up to 93 million people!)
            dI = binornd(meeting_events, infection_prob*susceptible
                         /(susceptible+infected));

            %Update population
            cities(n,3) = infected + dI;

            %Not more infected possible than whole population (due to
            %approximation this would be possible)
            if cities(n,3) > cities(n,2)
                cities(n,3) = cities(n,2);
            end
            %If infected are above 10000 in the city calculate without
            %a random part

        else
        %%
        %Calculate meeting events between infected and any other people
        %Normally distributed, rounded and abs taken
            meeting_events = round(abs(randn*meetings_stdev*infected +
                                   meetings_mean*infected));

           %Just take the mean of infections which would happen in this
           %case and round up. Approximation again, for higher
           %population maybe even more accurate.
            dI = ceil(meeting_events*infection_prob*susceptible
                      /(susceptible+infected));

            %Update population
            cities(n,3) = infected + dI;

            %Not more infected possible than whole population (due to
            %approximation this would be possible)
            if cities(n,3) > cities(n,2)
                cities(n,3) = cities(n,2);
            end
        end
    end
end
end
```

A.8 *transport_with_fixed_tot_T*

```
function cities = transport_with_fixed_tot_T(cities,edges,tot_T)
%% Function which calculates the transport of infected people
 N = cities(:,2); % Total population
I = cities(:,3); % Infected

% Transport matrix
% Col1: Infected voyagers going x -> y; Col2: Infected voyagers going y
-> x
T = zeros(length(edges(:,2)),2);

%Unsort edges matrix (is needed because otherwise allways the large
cities
%would calculate the travel first which could lead to artifacts.
for i = 1:length(edges)
    r(1) = ceil(rand*length(edges));
    r(2) = ceil(rand*length(edges));
    temp_edges = edges(r(1),:);
    temp_tot_T = tot_T(r(1));
    tot_T(r(1)) = tot_T(r(2));
    tot_T(r(2)) = temp_tot_T;
    edges(r(1),:) = edges(r(2),:);
    edges(r(2),:) = temp_edges;
end
```

```
%Run through all connections
for i = 1:length(edges)
    %%

    %For better overview define x as city 1 in the edges matrix and y
as
    %city 2 in the edges matrix
    x = edges(i,1);
    y = edges(i,2);

    %Only calculate the transport if there are any infected in the two
    %connected cities and if there are any susceptible
    if I(x) < N(x) && I(x) > 0

        %Hygestat calculates mean and variance for the hypergeometric
        %distribution. As a very fast approximation normaldistribution
        %with these parameters is used. Hypergeometric distributed
        %random numbers are extremely slow
        [mean, var] = hygestat(N(x),I(x),tot_T(i));

        %This calculates how many infected travel from x to y
        T(i,1) = abs(round(randn*sqrt(var) + mean));

        %hygestat cannot calculate hygestat(N(x),N(x),tot_T(i)) so this
        %has to be calculated seperately
    elseif I(x) == N(x)

        T(i,1) = tot_T(i);

    else

        T(i,1) = 0;

    end

    %Same as before but in the backwards direction
    if I(y) < N(y) && I(y) > 0

        [mean, var] = hygestat(N(y),I(y),tot_T(i));
        T(i,2) = abs(round(randn*sqrt(var) + mean));

    elseif I(y) == N(y)

        T(i,2) = tot_T(i);

    else

        T(i,2) = 0;

    end
    %Calculate number of infected in city x as: Infected(last tick) +
    %Infected travelling to this city - Infected travelling away from
    %this city
    I(x) = I(x) + T(i,2) - T(i,1);

    %Same for city y
    I(y) = I(y) + T(i,1) - T(i,2);

    %Because of the approximations more than total number of
    %inhabitants of a city can travel, which has to be corrected. Also
    %the total amount of infected in a city can become bigger than the
    %total number of inhabitants.
    if I(x) < 0
        I(y)= I(y) + I(x);
        I(x) = 0;
    end

    if I(y) < 0
        I(x)=I(x)+I(y);
        I(y) = 0;
    end

    if I(x) > N(x)
        I(y)=I(y)-I(x)+N(x);
        I(x)=N(x);
    end

    if I(y) > N(y)
        I(x)=I(x)-I(y)+N(y);
        I(y)=N(y);
    end

end

%Update cities matrix
cities(:,3) = I;

end
```

A.9 generate_output

```
function generate_output(output_array, tot_pop, cities, root,edges,b)
%% Function which generates the relevant output

%Generate b (counter in the parfor loop) as a string for later filename
bstr = int2str(b);

%%
%Calculate and produce file for number of cities with at least 10%
%infected with respect to the time. Additionally calculate the ratio of
%globlal infected to total population with respect to the time.

%Initialization
percent_10_infected(length(output_array(1,:)))=0;
ratio(length(output_array(1,:)))=0;
```

```
 %Go through all data points and allway calculate sums over all cities
for i = 1:length(output_array(1,:))
    percent_10_infected(i)=0;

    %Ratio contains the ratio of infected to total population
    ratio(i) = sum(output_array(:,i))/tot_pop;

    %Calculate the number of cities with at least 10% infected
    for g = 1:length(output_array(:,1))
        if output_array(g,i)/cities(g,2) >= 0.1
            percent_10_infected(i) = percent_10_infected(i) + 1;
        end
    end
end
%Create continuous filenames so no file will be overwritten when run on
%several computers. Then save files for the different experiments
percent_corr_name='percent_10_total_000.txt';
percent_corr_name(21-length(bstr):20)=bstr;
dlmwrite(percent_corr_name,percent_10_infected);
ratio_2nd_name='ratio_2nd_000.txt';
ratio_2nd_name(14-length(bstr):13)=bstr;
dlmwrite(ratio_2nd_name,ratio);
%For the distance correlation another output was used
% ---
% output_degree(1) = distance;
% output_degree(2) = t;
% degree_corr_name='degre5_corr000.txt';
% degree_corr_name(15-length(bstr):14)=bstr;
% dlmwrite(degree_corr_name,output_degree);
% ---
%For the degree correlation other values had to be calculated and the
%output was altered, too. With the function below the degree of the
%root node and the 1st generation connected nodes are calculated and
%taken as an output.
% ---
% connections = find(edges == root);
% tot_degree = cities(root,1);
% for i = 1:length(connections)
%     if connections(i) > length(edges)
%         connections(i) = connections(i) - length(edges);
%     end
%     if edges(connections(i),1) == root
%         working_node = edges(connections(i),2);
%     else
%         working_node = edges(connections(i),1);
%     end
%     tot_degree = tot_degree + cities(working_node,1);
% end
% output_degree(1) = tot_degree;
% output_degree(2) = t;
% degree_corr_name='degre7_corr000.txt';
% degree_corr_name(15-length(bstr):14)=bstr;
% dlmwrite(degree_corr_name,output_degree);
%---

end
```