Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Swiss Rail Network Formation with Physarum Polycephalum

Lucas Böttcher
Simon Roth
Gabriela Schär

Zurich
December 2012

Agreement for free-download

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

Lucas Böttcher   Simon Roth   Gabriela Schär

Contents

1. Abstract

The process of continuous urbanization is noticeable in Switzerland like in the most other countries.[1] Besides the benefits of combining educational, medical, cultural and industrial centers in a small are, there are also some problems arising. The cities are growing, whereupon this growth in population causes an increasing use of the public transport system, which has to be adapted to the new conditions. The main goal of this project is the simulation of the Swiss rail network depending on population growth. The network is simulated with a biological inspired model based on Physarum polycephalum. This slime mold is a large single-celled amoeboid organism that forages for food sources. To maximize the searched area, it explores its environment with a relatively continuous foraging margin. It is forming different junctions and nodes to reduce the overall length of the connecting network [7]. This principle is adapted to the main railroads in Switzerland.

2. Individual contributions

- Lucas Böttcher: Implementation Shortest Connection, Report
- Simon Roth: Implementation Swiss Rail Network Simulation, Report
- Gabriela Schär: Processing of geographic data, Report

3. Introduction and Motivations

Everyday thousands of commuters use the Swiss rail network to reach their workplace, school or university, mainly the same railroad is used. Seeing these facts we are going to ask, whether the actual situation of the network is the most efficient one. And related to that question, whether different connections between cities have maybe a higher performance.

We assume, that the rail network is developed in a way that the connections between cities are the most efficient. Certain biological organisms connect their food sources in a similar way. In this project we simulate a biological organism creating its own efficient connections between the cities. This approach leads to the following main questions:

1. Is the biological model a good approximation to simulate the network compared to reality?

[1]see the autumn sunday lectures at ETH Science City (Die Stadt der Zukunft - die Zukunft der Stadt)

2. Are there any new built or destroyed connections in the network because of the future population growth?

To answer these questions, the results of the simulations are compared with the Swiss rail network. For this comparison geographic data from geodata.ethz.ch [5] are used. This data are processed with *ArcGis 10.1* to get the basis for the simulation.

Cities are choosen based on the numbers of inhabitants (more than 10000) in 2011 and the presence of a railaway station. Additional to these some cities with less than 10000 inhabitants are added because of actual importance for the Swiss rail network [1].

To answer the second question, we are going to manipulate parameters in Physarum network, like described in 6.

We expect that the most efficient network created by the simulation is a good approximation of the actual Swiss railroad network. We also expect that the network will not change even when the population is growing because a network changes the most at the beginning of developing. And the Swiss railroad network reaches the end of development already.

4. Description of the Model

The model is inspired of the Physarum polycephalum and the way this single-celled fungus searching for food. The organism have been subjected to successive rounds of evolutionary selection and have found an appropriate balance between efficiency, cost and resilience. The plasmodium contains a network of tubes, which enables chemical signals and nutrients to circulate through the organism. If some of the tubes have found food sources, this implies a positive feedback to the system. This tubes are getting thicker so the flux of nutrients increase. Other tubes which do not connect to food sources are shrinking and tend to disappear, because there is no flux available. Experiments showed two empirical rules. **1)** Tubes with no connection to a source disappear and **2)** if there are two tubes connecting the same source, the longer one disappears. This led to useful approaches to problem-solving like optimization of railroad systems or networks in general.

The mathematical model is based on *Physarum solver: A biologically inspired method of road-network navigation* [6]. Figure 2 shows the concept of the mathematical model. There are different nodes. The first two nodes corresponding to the food sources (N_1 and N_2) and other nodes (N_3, N_4 ... N_j). The junction between the node N_i and N_j is denoted as M_{ij}. In the model we connect the single nodes with circular tubes and want to measure their inner flux (Figure 1). Therefore the fluid dynamics prepare us with the law of Hagen-Poiseuille, which allows us to calculate the flux from the difference in pressure, the tube length and some constants. Starting from the fluid velocity in a cylindric tube [3]:

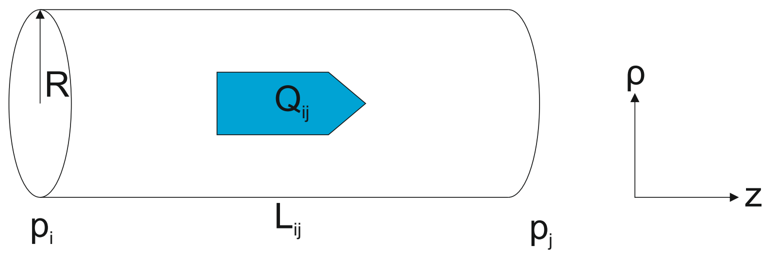

Figure 1: Illustration of the Hagen-Poiseuille law for a circular tube, which connects the single nodes in our model.

$$u(z) = -\frac{1}{4\eta}\frac{dp}{dz}(R^2 - \rho^2) \tag{1}$$

, where R is the radius of the cylinder and η the viscosity. Now we can integrate that expression over a circle, and normalize to get a mean velocity:

$$\overline{u(z)} = -\frac{1}{4\pi\eta R^2}\frac{dp}{dz}\int_0^{2\pi}\int_0^R (R^2 - \rho^2)\rho d\rho d\phi = -\frac{1}{8\eta}\frac{dp}{dz}R^2 \tag{2}$$

We can assume that the pressure gradient is uniform and express the mean velocity by the flux divided by its cross section:

$$Q_{ij} = \frac{D_{ij}}{L_{ij}}\Delta p_{ij} \tag{3}$$

, where $D_{ij} = -\frac{\pi R^4}{8\eta}$ is the conductance. So we see that the Hagen-Poiseuille law is related to Ohms law, where we have a similar expression.

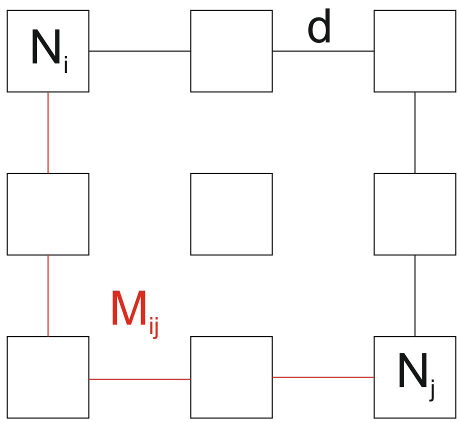

Figure 2: **a)** Concept of the mathematical model with a source and sink node. The distance between the given nodes is a constant d. The centered node is not reachable. **b)** Any square represents a node with red sources, yellow transport junctions (cyan is not used) and not reachable blue terrain.

The variable Q_{ij} stands for the flux through the junction M_{ij} from N_i to N_j. The flux Q_{ij} is given by an approximately Poiseuille flow where p_i is a pressure at the node N_i, L_{ij} is the length and D_{ij} is the conductivity of the junction M_{ij} :

$$Q_{ij} = \frac{D_{ij}}{L_{ij}} (p_i - p_j) \tag{4}$$

By considering Kirchhoff's law at each node, there is:

$$\sum_i Q_{ij} = 0 \quad \text{if } (j \neq 1, 2) \tag{5}$$

N_1 is assumed as a source node and N_2 as a sink and I_0 represent the flux from the source node and is in this model constant. It follows:

$$\sum_i Q_{i1} + I_0 = 0, \quad \sum_i Q_{i2} - I_0 = 0 \tag{6}$$

To describe the thickness of the junctions we assumed that the conductivity D_{ij} changes in time according to the flux Q_{ij}:

$$\frac{dD_{ij}}{dt} = f(\mid Q_{ij} \mid) - D_{ij} \tag{7}$$

where $f(\mid Q \mid)$ is a increasing function with $f(0) = 0$. Here $f(\mid Q \mid)$ is given by:

$$f(\mid Q \mid) = \frac{\mid Q \mid^{\gamma}}{1 + \mid Q \mid^{\gamma}} \tag{8}$$

The network Poisson equation for the pressure is derived from the equations (4), (5) and (6) as followed:

$$\sum_i \frac{D_{ij}}{L_{ij}} (p_i - p_j) = \begin{cases} -I_0 & \text{for } j = 1, \\ I_0 & \text{for } j = 2, \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

All p_i's can be determined by solving the equation system (9) when setting $p_2 = 0$ as a basic pressure level. With this also each Q_{ij} is obtained. They are definend by the D_{ij}'s and L_{ij}'s at each time step. Conductivity is closely related to the thickness of the junctions and so when a conductivity of a junction is zero, it disappears. The nodes N_1 and N_2 are randomly choosen for each time step.

5. Implementation

5.1. ArcGis

To get the basis matrix for the simulation, Swiss geographic data [5] are processed with *ArcGis 10.1*. Therefor the rasterdata are distinguished in different classes as in Table 1 are described.

Table 1: classes of data

Indentification	Class	Description
0	Ghostland	Cells in which the slime mold isn't allowed to be (foreign coutries, lakes)
1	Free land	Cells in which the slime mold is allowed to grow
2	Junction	Cells in which the slime mold is located, processed in MATLAB
3	City	Cells which represents food sources

All the geographic data are in vector format. So all lakes and the foreign countries can be erased. It would be possible to erase more types of covering (eg. rivers, rocks) where a train can not ride. The slope is in this case also neglected. So the mountains are not considerated in the simulation. Then all the choosen cities are imported to *ArcGis*. Therefor the coordinates (Y,X) [4] for each city are searched and a buffer of

2500m is layed on them. In this case no city will disappear when the vectordata get converted to rasterdata with a cellsize of 2500m. So after converting in rasterdata with the classes above, the basis matrix is exported as ASCII-File which can now used in MATLAB as simulation surface.

To compare the simulated results with the actual Swiss rail network, different types of railroads are neglected. Therefor every tunnel and every rail road which is not in use are not shown.

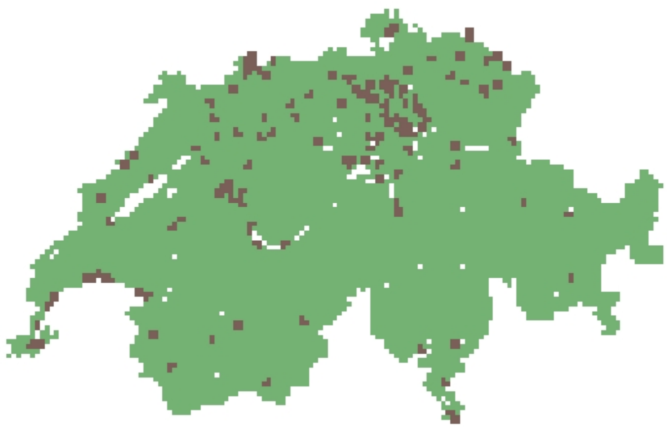

Figure 3: Rastardata from ArcGis with cellsize of 2500m - white: Ghostland, green: Free land, brown: City

5.2. MATLAB

5.2.1. Swiss Rail Network Simulation

To simulate the Swiss rail network a cellular automata approach with a von Neumann neighbourhood is used in this project because of the raster property of the model and the geogaphic data. This means, the cities are fixed nodes and all free land is populated with junctions. During the simulation junction cells with low conductivity are changed to free land. The basic structure of the network simulation is described in Figure 4.

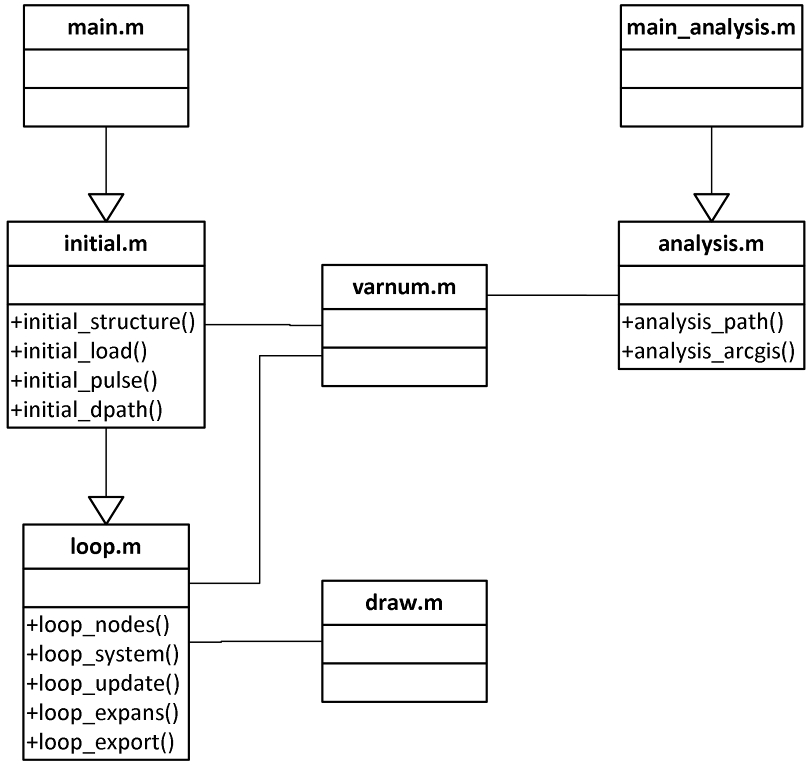

Figure 4: Structure of the network simulation implementation in MATLAB.

The simulation can be separated in two tasks, first the computation of data and second the analysis of data. Because of the long duration of one simulation and the lack of a break condition, owning to quasi-stationary conditions, the two task are run separatly.

The **main.m** script holds all necessary informations like parameters, constants, folder and file paths to run the simulation. A backup function is implemented, which can be turned on an off, because of program crashes during testing. The three input files geodata.txt, population.txt and growth.txt, which have to be in an appropriate form without header row, are used to run the simulation. All input data are processed to input variables for the main loop in the **initial.m** function. This function initializes the folder structure of the simulation, converts the input data, restores the backup if turned on and passes all the data to the main loop. The main tasks of the **loop.m** function are to compute the linear equation system (equation 9) and to solve the equations. For the equation system a index for every node and junction cell has to be computed. Due to performance loss by using the built in sub2ind MATLAB function the **varnum.m** function is implemented, which returns the index from a cell depending on the coordinates. After solving the linear equation system the new conductivities and an updated path can be computed. Because of the machine precision (eps) a junction cell already disappears before the conductivity is zero. This is implemented after noticing this behaviour in program tests. The simulation of the Swiss rail network needs a lot of time depending on the size of the input data, so a output file is implemented to control the progress and already computed results. To make a decision about the ending of the simulation a visual break conditions is implemented to see how far the network is. This picture is saved by the **draw.m** function and displays the actual network.

The **main_analysis.m** script contains all the variables to compute the final simulation results, which are computed in the **analysis.m** function. The final network and overall conductivity is converted to ASCII files, which are used to compare the different scenarios in *ArcGIS*. To analyse the development of the path length over time these two variables are stored in vectors to be plotted afterwards.

The whole program is attached to the appendix A.1.

5.2.2. Shortest Connection

For evaluating the path length given by the Physarum simulation data, a program is used, which computes the shortest path length between two given points on the map. A standard algorithm [2] for implementing the function in MATLAB is used. In general the code uses a start coordinate from an input file, related to the map matrix A with the defined neighborhood, i.e.:

```
function [B,short_length] = shortest_path(A,points,neigh)
```

By calling the function with the necessary arguments a matrix B is got, which contains the given map, without citites or earlier computed ways, but with the drawn shortest path. The shortest path length **int** is a second output argument of the function. The algorithm explained in a nutshell can be described with the following words: After cleaning the computed matrix from cities and ways, the start entry is coded with two. As a result the matrix contains only zeros (ghostland), ones (free land) and the start entry two. The algorithm iterats over the whole matrix entries and continues in the loop. If the entry is not one, the algorithm jumps to the next cell:

```
if B(k,l) ~= 1
    continue;
end
```

The cells are numerated starting by two and the next reachable cell gets the next natural number. This iterates with the given neighborhood:

```
if B(m,n) == step+1
    B(k,l) = step+2;
    iter = 1;
end
```

The variable **iter** shows if there is any neighbour (=1), if not the loops stops:

```
if ~iter
    break;
end
```

At the end the output is a numerated path, which can be used to calculate the track length or to use a backtracking algorithm for getting the path in the output matrix B. The whole program is attached to the appendix A.2.

6. Simulation Results and Discussion

All the simulations are ran with the same parameters and with a cellsize of 2500m.

In Figure 5 and Figure 7 are shown the results of the first MATLAB simulation and the actual rail network. In this simulation is assumed that every city has an equal number of inhabitants. The choosen path is in Figure 5 ploted. Under consideration of the choosen cities and the neglected slope this simulated path is a good approximation of the real rail network. The plot in Figure 6 showes the developement of the overall path length over time. At first the function is on a constant level and then the length decreases in an exponential way. With more time steps, the function has to itterate over less cells and so less time steps are needed to proceed. Around time step 40 the simulation tries to get in a stady-state but a little change prevents it reaching an equilibrium.

In Figure 7 is shown the conductivity which goes along with this simulated path. It describes which line is used the most (e.i. Bern-Solothurn-Sursee-Lucern-Zug-Zurich).

This first simulation is ran twice, each time with the same result.

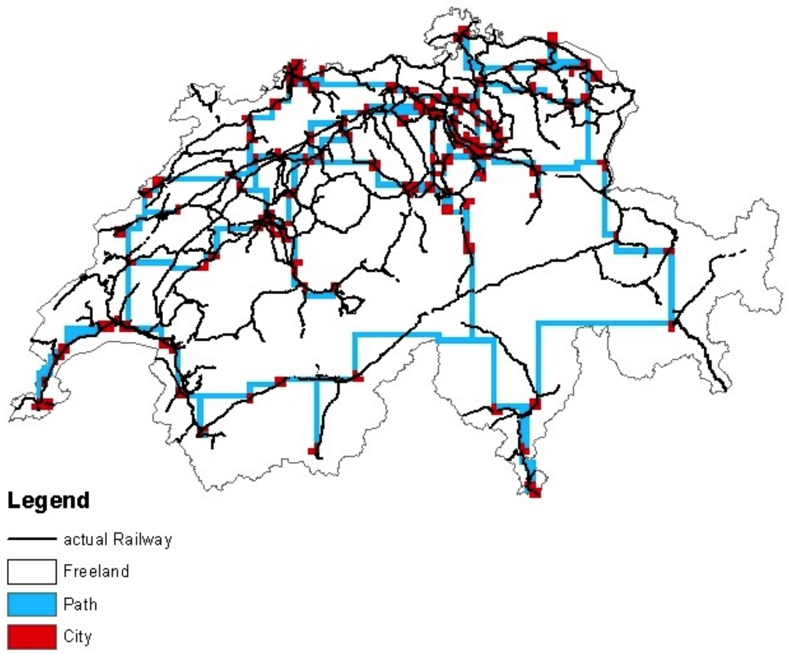

Figure 5: Simulated path of Physarum polycephalum and the actual Swiss rail network

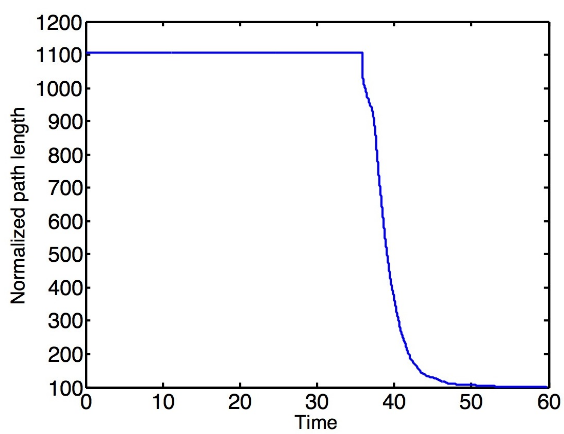

Figure 6: Developement of the overall path length according to Figure 5; path length in normalized numbers of cells containing slime mold

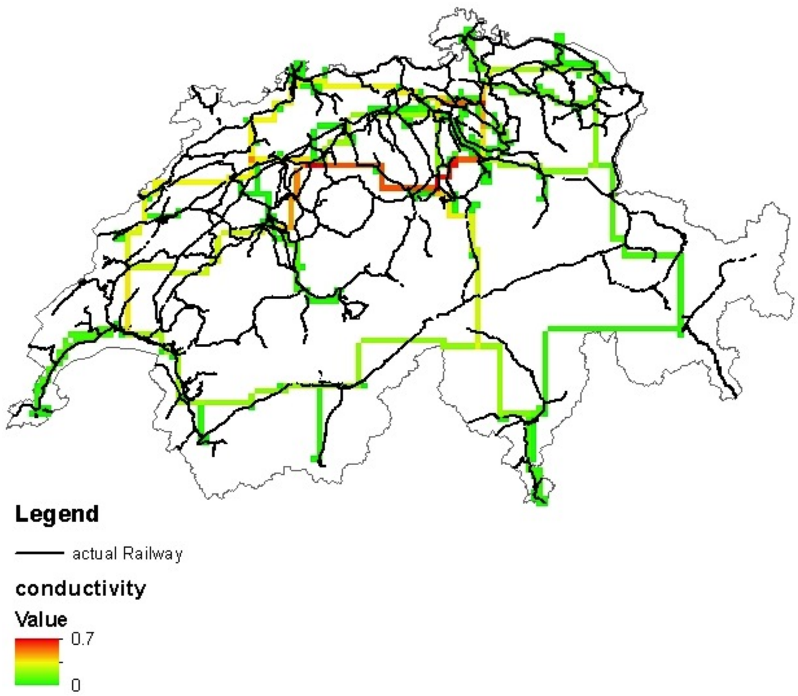

Figure 7: Simulated conductivity of Physarum polycephalum and the actual Swiss rail network

To answer the second main question, a second simulation is ran. This time, each city has its own number of inhabitants corresponding to the assumend population in 2050 calculated with a linear growth [1]. The result is shown in Figure 8. Comparing to the actual path there are just slight differences. Except of the junction between St. Moritz-Bellinzona and Bern-Solothurn, which disappear in the simulation of 2050. This could be explained with the low population of both of these cities (St. Moritz/Bellinzona) and in this case the connection gets less important. The disappeared connection between Bern and Solothurn can be explained with different numbers of time steps over which the simulation itterates (first simulation: 5973 time steps, second simulation: 7318 time steps). With more time steps the junction would also disappear in the first simulation.

Comparing the plots of the conductivity there are just slight differences too. In general, the conductivity is lower than in the first simulation. That depends on lower initial flux caused by the different populations, which is higher in the first simulation. The line with the highest conductivity is longer in the simulation of 2050 than in the first simulation as shown in Figure 10. The most frequently used line goes now from Lausanne over Bern, Solothurn, Lucern, Zug to Zurich. This leads to the conclusion that the Swiss rail network is not going to change in the future.

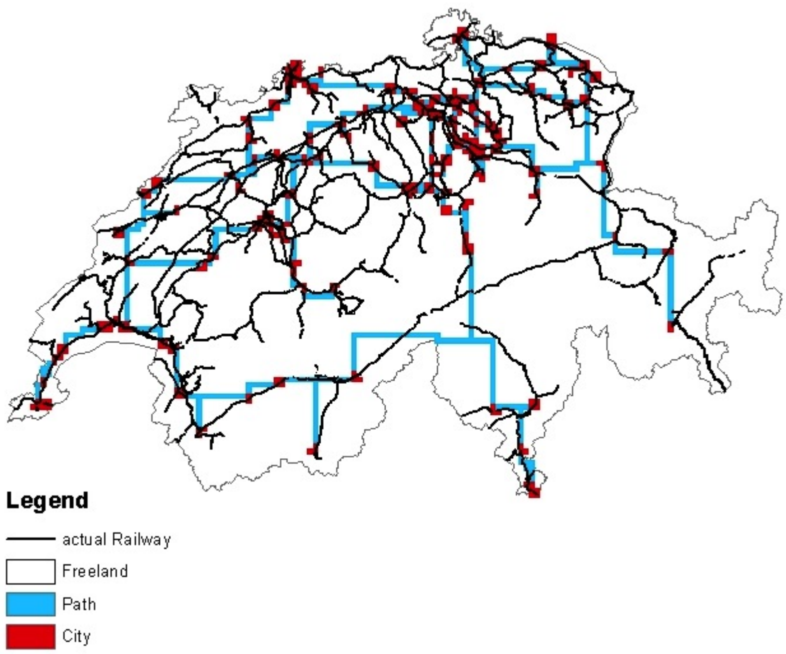

Figure 8: Simulated path in 2050 of Physarum polycephalum and the actual Swiss rail network

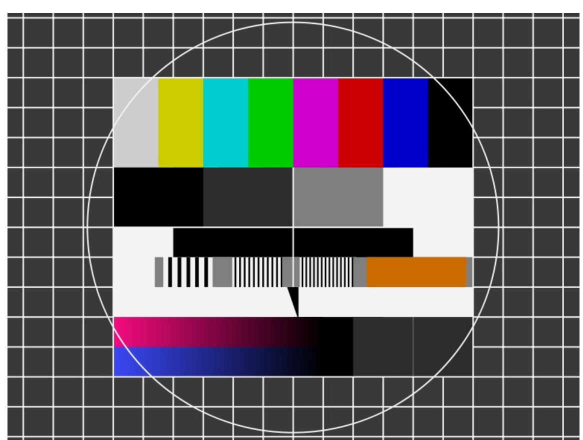

Figure 9: Developement of the overall path length according to Figure 8; path length in normalized numbers of cells containing slime mold

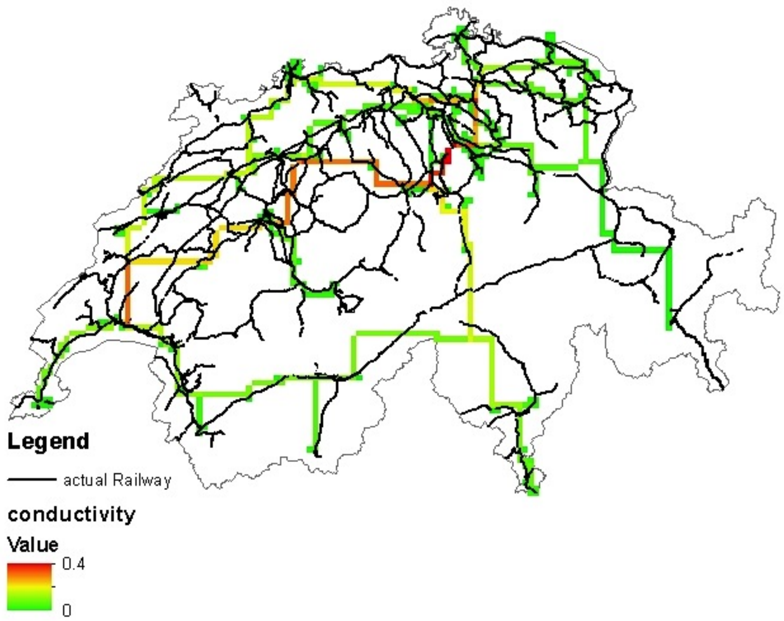

Figure 10: Simulated conductivity in 2050 of Physarum polycephalum and the actual Swiss rail network

The question comes up, if the simulated path of the slime mold is the shortest connection between two cities. To verify this assumption a second program is used (compare 5.2.2). The result is that the lengths between the cities are in both simulations equal as shown in Figure 11. This make sense, because the shortest path is often the one with the highest efficiency.

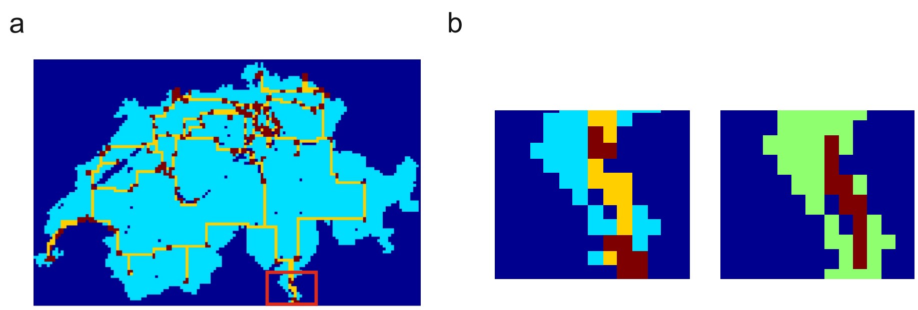

Figure 11: **a)** Simulated path length from the Physarum simulation code. **b)** Zooming into the red square is on the left hand side the Physarum simulation path length, compared to the right hand side computed shortest path length shown.

7. Summary and Outlook

The main goal of this project was to simulate the Swiss rail network based on a biological inspired model. In a first simulation the path of the Swiss rail network is simulated and in a second simulation the influence on the network based on the population in the different cities is investigated. So both main questions can be answered.

1. The simulation leads to a good approximation of the real Swiss rail network under consideration that different surface covers and the slope is neglected.

2. Depending on the population growth there are built no new connections but one is disappeared.

The model based on Physarum polycephalum leads in general to a good approximation of a rail network. But the utilization of a rail network does not only depend on the number of people living in a city. In this project, this is the only parameter which the model uses to decide which connection is efficient. There are also other impacts which make a specific connection more efficient or more attractiv to use. For example how fast can a train ride and following this how long lasts a journey. Especially in the Alps the number of tourists and the touristic villages with less inhabitants are an important factor.

There are different points to continue with this project or maybe to enlarge it in the future. With more time and hardware capacity it is possible to do a lot more simulations to find the best set of parameters. Another interessting point would be to simulate the rail network with more restrictions like the slope or other surface coverings. It is also possible to simulate different population growth scenarios to see the consequences in the resulting rail network. Another way to use the model is to preset the actual rail network and to simulate the utilization of the different rail lines. Further could the international connections be considered.

A. Matlab-Code

A.1. Network Simulation

A.1.1. main.m

```
% Modeling and Simulating Social Systems with MATLAB
% http://www.soms.ethz.ch/matlab
% Author: Gabriela Schaer, Simon Roth, Lucas Boettcher 2012
% http://github.org/Trail—Formation


clear


% parameters and constants
%———————————————————————————————————————————————————————————————————————————

D0 =   1.00;     %initial conductivity
I0 = [ 1; 3 ];  %flux interval
g  =   1.80;     %expans equation gamma
a0 =   2011;     %year of simulation data
a  =   2050;     %year of simulation, no growth for a < a0

T  = 1000;       %end of simulation
dt = 0.01;       %timestep of simulation


% backup simulation
%———————————————————————————————————————————————————————————————————————————

backup    = 0;               %1 for TRUE, 0 for FALSE
simfolder = '';              %folder of simulation to backup (with slash /)


% folder sturcture
%———————————————————————————————————————————————————————————————————————————

import = 'input/';       %folder of input data
export = 'output/';      %folder to save the simulation results


% input data
%———————————————————————————————————————————————————————————————————————————

geofile   = 'geodata.txt';        %coded geodata
popfile   = 'population.txt';     %population
growfile = 'growth.txt';         %population growth in percent
```

```
% initialize simulation
%-----------------------------------------------------------------------------

initial( D0, I0, g, a0, a, T, dt, backup, simfolder, import, export, geofile, popfile, growfile )
```

A.1.2. initial.m

```
function [] = initial( D0, I0, g, a0, a, T, dt, backup, simfolder, import, export, geofile, popfi
%INITIAL prepars all the input data for the simulation.
%   All parameters and constants and all input data are converted and
%   prepared to run the simulation.

    % folder structure
    simpath = initial_structure( backup, simfolder, export );

    %load input data
    [ geo, pop, grow ] = initial_load( backup, import, geofile, popfile, growfile, simpath );

    % get dimension of geodata
    [ Y, X ] = size( geo );

    % get indices of nodes
    nodes = find( geo == 3 );

    % get pulses
    I = initial_pulse( I0, a0, a, pop, grow, nodes );

    % define von Neumann neighbourhood (x y)
    neigh = [         0 -1;
              -1  0;          1  0;
                      0  1       ];

    % initialize conductivity and path
    D    = zeros( X*Y, X*Y );
    path = geo;

    [ D, path, t0 ] = initial_dpath( dt, backup, D0, simpath, geo, Y, X, neigh, D, path );
```

```
    % initialize flux
    flux = zeros( X*Y, X*Y );


    % export data for backup
    if ( backup ~= 1 )

        save( [ simpath, 'geodata' ], 'geo' );
        save( [ simpath, 'population' ], 'pop' );
        save( [ simpath, 'growth' ], 'grow' );

    end


    % start main loop
    loop( g, a0, a, T, dt, geo, simpath, Y, X, nodes, I, neigh, D, path, t0, flux );


end


%% create folder structure
function [ simpath ] = initial_structure( backup, simfolder, export )


    if ( backup == 1 )

        %folder path
        simpath = [ export, simfolder ];

    else

        %get unique foldername
        foldername = int2str( now*10^5 );

        %create folders
        mkdir( [ export, foldername ] );
        mkdir( [ export, foldername, '/conductivity' ] );
        mkdir( [ export, foldername, '/path' ] );

        %folder path
        simpath = [ export, foldername, '/' ];

    end


end
```

```
%% load input data
function [ geo, pop, grow ] = initial_load( backup, import, geofile, popfile, growfile, simpath )


    if ( backup == 1 )

        % load exported files
        load( [ simpath, 'geodata.mat' ] );
        load( [ simpath, 'population.mat' ] );
        load( [ simpath, 'growth.mat' ] );

    else

        % load input files
        geo  = load( [ import , geofile ] );
        pop  = load( [ import, popfile ] );
        grow = load( [ import, growfile ] );

    end


end


%% initialize pulses
function [ I ] = initial_pulse( I0, a0, a, pop, grow, nodes )


    % initial pulses with mean of flux interval
    I = ones( length( nodes ), 1 )*( ( I0( 1 )+I0( 2 ) )/2 );

    % compute pulses for population growth
    if ( a > a0 )

        for i = 1:length( nodes )

            % assume a linear growth
            I( i ) = pop( nodes( i ) )*( 1+grow( nodes( i ) )/100 )^( a-a0 );

        end

        % compute linear function of fluxes in interval
        a = ( I0( 2 )-I0( 1 ) )/( max( I )-min( I ) );
        b = 1 - ( I0( 2 )-I0( 1 ) )/( max( I )-min( I ) )*min( I );

        % final flux matrix
        I = a*I+b;

    end
```

```
end


%% initialize conductivity and path
function [ D, path, t0 ] = initial_dpath( dt, backup, D0, simpath, geo, Y, X, neigh, D, path )


    % initial starttime
    t0 = 0;

    if ( backup == 1 )

        %get all saved conductivities
        backupD = dir( [ simpath, 'conductivity_*.mat' ] );

        %get last conductivity
        lastD = backupD( length( backupD ) ).name;

        %load last conductivity
        load( [ simpath, lastD ] );

        %get last timestep
        t0 = sscanf( lastD, 'conductivity_%i.mat' )*dt+dt;

    end

    % initialize or restore conductivity matrix
    for i = 1:X
        for j = 1:Y

            for k = 1:length( neigh )

                m = i+neigh( k, 1 );
                n = j+neigh( k, 2 );

                if ( geo( j, i ) > 0 && m >= 1 && n >= 1 && m <= X && n <= Y && geo( n, m ) > 0 )

                    p = varnum( n, m, Y );
                    q = varnum( j, i, Y );

                    if ( backup ~= 1 )

                        D( p, q ) = D0;

                    end

                    if ( geo( p ) ~= 3 && D( p, q ) > eps )
```

```
                        path( p ) = 2;

                    end

                end

            end

        end
    end


end
```

A.1.3. loop.m

```
function [] = loop( g, a0, a, T, dt, geo, simpath, Y, X, nodes, I, neigh, D, path, t0, flux )
%LOOP is the main loop of simulation.
%   This function is the main loop of the simulation. After the
%   initialisation, every step is controled from here.


    %initialize loop history variables
    oldpath = 0;

    for t = t0:dt:T

        % initialize equation system
        A = zeros( X*Y, X*Y );
        b = zeros( X*Y, 1 );


        % set source and sink node
        [ soury, sourx, sinky, sinkx, curI ] = loop_nodes( T, Y, X, nodes, I );


        % get equation system
        [ A, b ] = loop_system( geo, Y, X, neigh, D, path, A, b, soury, sourx, sinky, sinkx, curI


        % solve equation system
        press = A\b;


        % update conductivity and path
        [ D, path ] = loop_update( g, dt, geo, Y, X, neigh, D, flux, press );
```

```
        % compute and save results
        newpath = sum( find( path == 2 ) );

        loop_export( dt, simpath, D, path, oldpath, a0, a, t, newpath );

        oldpath = newpath;

    end


end


%% get random source and sink node
function [ soury, sourx, sinky, sinkx, curI ] = loop_nodes( T, Y, X, nodes, I )


    for u = 0:T

        % random node numbers
        v1 = randi( length( nodes ), 1 );
        v2 = randi( length( nodes ), 1 );

        if ( v1 ~= v2 )

            break

        end

    end

    % get source and sink coordinates
    [ soury, sourx ] = ind2sub( [ Y, X ], nodes( v1 ) );
    [ sinky, sinkx ] = ind2sub( [ Y, X ], nodes( v2 ) );

    % get flux of source
    curI = I( v1 );


end


%% compute equation system
function [ A, b ] = loop_system( geo, Y, X, neigh, D, path, A, b, soury, sourx, sinky, sinkx, cur


    for i = 1:X
        for j = 1:Y
```

```
            for k = 1:length( neigh )

                m = i+neigh( k, 1 );
                n = j+neigh( k, 2 );

                if ( m >= 1 && n >= 1 && m <= X && n <= Y )

                    p = varnum( n, m, Y );
                    q = varnum( j, i, Y );

                    if ( i == sourx && j == soury )

                        b( q ) = -curI;

                    end

                    if ( i == sinkx && j == sinky || path( j, i ) < 2 )

                        A( q, q ) = 1;

                    elseif ( geo( j, i ) > 0 && geo( n, m ) > 0 )

                        A( q, p ) = A( q, p )+D( p, q );
                        A( q, q ) = A( q, q )-D( p, q );

                    end

                end

            end

        end
    end


end


%% compute new conductivity and set path
function [ D, path ] = loop_update( g, dt, geo, Y, X, neigh, D, flux, press )


    % reset path
    path = geo;

    for i = 1:X
        for j = 1:Y

            for k = 1:length( neigh )
```

```
                m = i+neigh( k, 1 );
                n = j+neigh( k, 2 );

                if ( geo( j, i ) > 0 && m >= 1 && n >= 1 && m <= X && n <= Y && geo( n, m ) > 0 )

                    p = varnum( n, m, Y );
                    q = varnum( j, i, Y );

                    flux( p, q ) = D( p, q )*( press( p )-press( q ) );
                    D( p, q )    = D( p, q )+( loop_expans( g, flux( p, q ) )-D( p, q ) )*dt;

                    if ( geo( n, m ) ~= 3 && D( p, q ) > eps )

                        path( n, m ) = 2;

                    end

                end

            end

        end
    end


end


%% tube expansion function
function [ fQ ] = loop_expans( g, Q )


    fQ = abs( Q )^g/( 1+abs( Q )^g );


end


%% save the results
function [] = loop_export( dt, simpath, D, path, oldpath, a0, a, t, newpath )


    % only save results on change
    if ( newpath ~= oldpath )

        draw( path , 'jet', [ simpath, 'path/' ], [ 'path_', num2str( 1/dt*t ) ] );
        save( [ simpath, 'conductivity/', 'conductivity_', num2str( 1/dt*t ) ], 'D' );

    end
```

```
    % update state file
    save( [ simpath, 'state.txt' ], 'a0', 'a', 't', 'newpath', '-ascii' );


end
```

A.1.4. varnum.m

```
function [ num ] = varnum( y, x, Y )
%VARNUM assigns a unique number to every node and junction.
%   A unique number for every node or junction is nessecary to generate
%   and solve the equation system. The input variables are the coordinates
%   x y and the X dimension of the simulation surface.

    num = ( x-1 )*Y+y;

end
```

A.1.5. draw.m

```
function [] = draw( data, color, folderpath, filename )
%DRAW exports graphic of current simulation.
%   Automatic graphic export with input parameters to adjust appearance.


    set( gcf, 'Position', [ 0 0 1 1 ] );
    imagesc( data )
    colormap( color )
    axis image
    axis off

    saveas( gcf, [ folderpath, filename,'.png' ] );


end
```

A.1.6. main_analysis.m

```
% Modeling and Simulating Social Systems with MATLAB
% http://www.soms.ethz.ch/matlab
% Author: Gabriela Schaer, Simon Roth, Lucas Boettcher 2012
% http://github.org/Trail-Formation


clear


% simulation data
%--------------------------------------------------------------------------

simpath  = 'output/73520551715/';
dt       = 0.01;

load( [ simpath, 'geodata.mat' ] );
cellsize = 2500;                        %[m]


% start analysis
%--------------------------------------------------------------------------

analysis( simpath, dt, geo, cellsize );
```

A.1.7. analysis.m

```
function [] = analysis( simpath, dt, geo, cellsize )
%ANALYSIS computes all results of the simulation
%   All simulation data is processed and stored in appropriate form for
%   further use.


    % create analysis folder
    mkdir( [ simpath, 'analysis' ] );
    analysispath = [ simpath, 'analysis/' ];


    % analyse path
    analysis_path( simpath, dt, geo, analysispath );


    % convert to arcGIS
    analysis_arcgis( analysispath, cellsize );
```

```
%% analyse path
function [ D ] = analysis_path( simpath, dt, geo, analysispath )


    % get dimensions
    [ Y, X ] = size( geo );

    % define von Neumann neighbourhood (x y)
    neigh = [         0 -1;
             -1  0;          1  0;
                      0  1       ];

    % list all conductivity files
    conductivity = dir( [ simpath, 'conductivity/conductivity_*.mat' ] );
    S = length( conductivity );

    % get enttime of simulation
    lastD = conductivity( S ).name;
    T = sscanf( lastD, 'conductivity_%i.mat' );

    % initialize path over time vectors
    timepath = zeros( 1, T+1 );
    time     = zeros( 1, T+1 );

    % initialize waitbar
    w = waitbar( 0, 'Please wait...' );

    for t = 0:T

        for s = 1:S

            % get current conductivity
            currentD = conductivity( s ).name;

            % compute on change
            if ( sscanf( currentD, 'conductivity_%i.mat' ) == t )

                % reset path
                path = geo;

                loadD = conductivity( s ).name;
                load( [ simpath, 'conductivity/' loadD ] );

                % initialize summarized conductivity
                sumD = zeros( Y, X );

                for i = 1:X
                    for j = 1:Y
```

```
                    for k = 1:length( neigh )

                        m = i+neigh( k, 1 );
                        n = j+neigh( k, 2 );

                        if ( geo( j, i ) > 0 && m >= 1 && n >= 1 && m <= X && n <= Y && g

                            p = varnum( n, m, Y );
                            q = varnum( j, i, Y );

                            if ( geo( p ) ~= 3 && D( p, q ) > eps )

                                path( p ) = 2;

                            end

                            if ( s == S && path( q ) > 1 )

                                sumD( q ) = sumD( q )+D( p, q );

                            end

                        end

                    end

                end
            end

            pathlength = sum( find( path == 2 ) );

            break

        end

    end

    % vectors for plot
    time( t+1 )     = t*dt;
    timepath( t+1 ) = pathlength;

    % saver vectors
    save( [ analysispath, 'time' ], 'time' );
    save( [ analysispath, 'timepath' ], 'timepath' );

    waitbar( t/T );

end
```

```
    % close waitbar
    close( w );

    % export matrix files
    save( [ analysispath, 'path' ], 'path' );
    save( [ analysispath, 'conductivity' ], 'sumD' );

end


%% arcGIS conversion
function [ path, sumD ] = analysis_arcgis( analysispath, cellsize )


    % convert path
    load( [ analysispath, 'path.mat' ] );
    [ nrows, ncols ] = size( path );

    file = fopen( [ analysispath, 'path.txt' ], 'w' );

    % header of ascii file
    header = { 'ncols         ', ncols;
               'nrows         ', nrows;
               'xllcorner     ', 0;
               'yllcorner     ', 0;
               'cellsize      ', cellsize;
               'NODATA_value  ', 0};

    for row = 1:size( header, 1 )

        fprintf( file, '%s%d\n', header{ row, : } );

    end

    fclose( file );

    dlmwrite( [ analysispath, 'path.txt' ], path, '-append', 'delimiter', ' ' );


    % convert conductivity
    load( [ analysispath, 'conductivity.mat' ] );
    [ nrows, ncols ] = size( sumD );

    file = fopen( [ analysispath, 'conductivity.txt' ], 'w' );

    % header of ascii file
    header = { 'ncols         ', ncols;
               'nrows         ', nrows;
               'xllcorner     ', 0;
               'yllcorner     ', 0;
```

```
                 'cellsize     ', cellsize;
                 'NODATA_value ', 0};

        for row = 1:size( header, 1 )

            fprintf( file, '%s%d\n', header{ row, : } );

        end

        fclose( file );

        dlmwrite( [ analysispath, 'conductivity.txt' ], sumD, '-append', 'delimiter', ' ' );

    end

end
```

A.2. Shortest Path

```
% Modeling and Simulating Social Systems with MATLAB
% http://www.soms.ethz.ch/matlab
% Author: Gabriela Schaer, Simon Roth, Lucas Boettcher 2012
% http://github.org/Trail-Formation

%PRE: A is the map with start: 2, free and end: 1, ghost 0, start and end
%coordinates from points vector and the neighborhood from neigh

%POST: Matrix B with drawn shortest path and the length (short_length)

function [B,short_length] = shortest_path(A,points,neigh)

% This function computes one shortest path and whose length between
%two points given in a 1x4 matrix, with coordinates related to the
%quadratic matrix A

% read matrix dimension
ALength1 = size(A,1);
ALength2 = size(A,2);

%initialize matrix B
B=A;

%Clear matrix B from ways and cities
```

```
for i=1:ALength1
    for j=1:ALength2
        if B(i,j)==2 || B(i,j)==3
            B(i,j)=1;
        end
    end
end

% iterate over the given points

    B(points(1),points(2)) = 2;

    % iterate over grid matrix A
            for step=1:ALength2^2

                % neighbor watch
                iter = 0;

                for k=1:ALength1
                    for l=1:ALength2

                        if B(k,l) ~= 1
                            continue;
                        end

                        % we use the linked neighborhood

                        for p = 1:size( neigh, 1 )

                        m = k+neigh( p, 1 );
                        n = l+neigh( p, 2 );

                        % need to respect borders

                            if m >= 1 && n >= 1 && m <= ALength1 && n <= ALength2

                                if B(m,n) == step+1
                                B(k,l) = step+2;
                                iter = 1;
                                end

                            end

                        end
                    end

                end
                % if there is no other neighbor, break
                    if ~iter
                        break;
```

```
                end

            end

        %compute the length from the marked elements in A
        short_length = B(points(3),points(4))-B(points(1),points(2));

        %initialize the jumping backward iteration points (for the
        %backtracking drawing of the way)
        end1=points(3);
        end2=points(4);
    for k=1:short_length

        for p = 1:size( neigh, 1 )

                        m = end1+neigh( p, 1 );
                        n = end2+neigh( p, 2 );

                        % need to respect borders

                            if m >= 1 && n >= 1 && m <= ALength1 && n <= ALength2

                                if B(m,n) == B(end1,end2)-1;
                                B(end1,end2) = 2;
                                end1=m;
                                end2=n;
                                end

                            end
                            continue;
        end


    end

     for k=1:ALength1
     for l=1:ALength2
         if B(k,l)~=2 && B(k,l)~=1 && B(k,l)~=0
             B(k,l)=1;
         end
     end
     end
end
```

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

Swiss Rail Network Formation with Physarum Polycephalum

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

Author(s)

Last name	First name
Böttcher	Lucas
Roth	Simon
Schär	Gabriela

Supervising lecturer

Last name	First name
Balietti	Stefano
Donnay	Karsten

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.

Zurich, December 2012
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

References

[1] Bilanz der ständigen Wohnbevölkerung (Total) nach Bezirken und Gemeinden. http://www.bfs.admin.ch/bfs/portal/de/index/themen/01/02/blank/data/01.html, October 2012.

[2] Bernd Gärtner. Informatik für Mathematiker und PhysikerInformatik für Mathematiker und Physiker. http://www.ti.inf.ethz.ch/ew/courses, November 2012.

[3] Brian Kirby. *Micro- and Nanoscale Fluid Mechanics*. Cornell University, New York, October 2010.

[4] Koordinator. http://tools.retorte.ch/map/, October 2012.

[5] GeoVITe ETH Geodata Portal. http://www.geodata.ethz.ch, October 2012.

[6] Atsushi Tero, Ryo Kobayashi, and Toshiyuki Nakagaki. Physarum solver: A biologically inspired method of road-network navigation. *Physica A Statisticla Mechanics and its Applications*, 2006.

[7] Atsushi Tero, Seiji Takagi, Tetsu Saigusa, Kentaro Ito, Dan P. Bebber, Mark D. Fricker, Kenjo Yumiki, Ryo Kobayashi, and Toshiyuki Nakagaki. Rules for biologically inspired adaptive network design. *Science Magazine*, 2010.