# Network models of epidemics

**Level 2 module in "Modelling course in population and evolutionary biology"**

**(701-1418-00)**

Module author: Viktor Müller

Course director: Sebastian Bonhoeffer
Theoretical Biology
Institute of Integrative Biology
ETH Zürich

## 1 Introduction

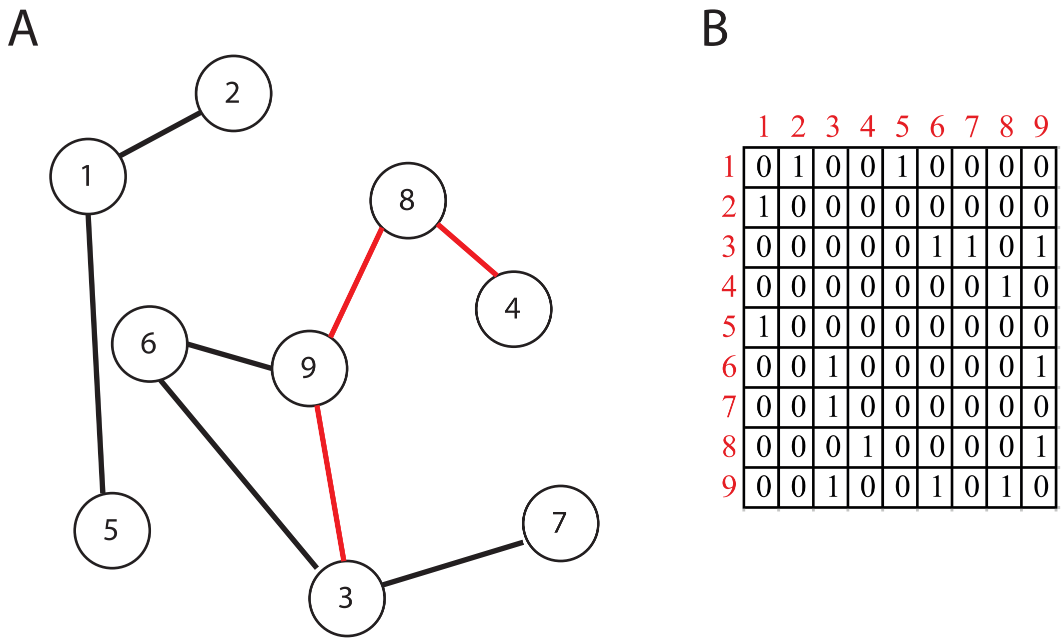
The classical models of epidemics (as described in the module 'SIR models of epidemics') have been constructed as sets of ordinary differential equations (ODEs), which implies several simplifying assumptions: time and population sizes change on continuous scales, with all processes occurring continuously and simultaneously; there is complete mixing within the classes or compartments of the models; and a given set of initial conditions always lead to exactly the same outcome. Real biological systems tend to violate these assumptions: populations consist of individuals and therefore population size can only change in discrete steps (on integer scale) and at discrete events (birth, death, etc.); initial conditions can never be defined perfectly, and unknown or uncontrolled variation results in the variability (stochasticity) of the outcome; and, finally, not all individuals can interact with each other. More complex models can remedy each of these shortcomings. Simple stochastic models (as in the module 'Stochastic simulation of epidemics') can deal with discrete scales (with respect to both time and population size), and can generate a range of distinct outcomes by implementing stochastic (random) effects. However, these models still regard all individuals as "equal" and interchangeable, implying that any two individuals can interact with each other with the same probability. This is essentially never true in real systems. Most interactions, and the transmission of most diseases, require proximity or at least become weaker or less likely with increasing distance. In populations of mobile individuals (e.g. humans) the patterns of proximity can best be traced by tracking the network of contacts between the individuals. In this module, we will develop simulation models that do this. Such models belong to the class of "individual-based models", because they track each individual separately, which allows the individuals to vary from each other, to form interactions according to specific rules, and to accumulate a "history" over the course of a simulation that can affect their subsequent behaviour. This is the most detailed and flexible of all modelling frameworks.

Below we will go through the basics of network theory, then link it to the study of epidemic spreading. Before going into that, however, let me address a general problem of modelling. If individual-based modelling is so much more powerful than other methods, then why bother with simple models at all? The answer is two-fold. First, for the sake of economy. Simpler models are easier and faster to construct and use less resources (e.g. computer time). Second, simpler models can actually be more powerful than their complex counterparts in one important respect: they are easier to understand and analyze. For ODE models, we can typically derive the complete range of possible behaviours and easily assess the effect of individual factors. In individual-based models this is a far more demanding task and it is often even difficult to decide whether we have actually accomplished it. Therefore, as a rule of thumb, the optimal modelling strategy is to use the *minimum level of complexity* required to yield appropriate answers to our questions. However, the limitations of the simpler models can only be defined by testing more complex models, as well. A major motivation for using complex models is to assess when and how complexity affects the behaviour of the models. In this module, we will investigate when and how network structure affects the behaviour of epidemics.

## 1.1 Network theory

Network theory is based on the application of graphs to real-world phenomena. A graph is an abstract representation of a set of objects where some pairs of the objects are connected by links. In graph theory, the objects can be called *nodes* or *vertices*, and the links can also be called *edges*. In epidemiology, you can simply refer to *individuals* and their *contacts*. Here we will mostly deal with simple undirected graphs: Figure 1 shows such an example. Nodes/individuals are labelled with numbers and the lines between them indicate (mutual/symmetrical) contacts. There is a *path* between two nodes if one can be reached from the other by traversing through a series of adjoining links. In the example, the shortest path between Nodes 3 and 4 has length 3 and is highlighted in red. This graph consists of two *connected components*: all nodes of a connected component (e.g. Nodes 1, 5, 2) can be reached from any other node of the component; however, there are no links between the two components. The *degree* of a node in an undirected graph is the number of links connecting to it, e.g. Node 1 has degree 2, and Node 9 has degree 3 in the example.

There are a number of ways to define and store a graph in `R` (or elsewhere). Two of the most straightforward methods are the *adjacency matrix* and the listing of all edges. For a graph with $n$ nodes, the adjacency matrix is a square matrix with dimension $n \times n$ such that $A_{i,j} = 1$ if nodes $i$ and $j$ are connected, and zero if they are not. For undirected graphs $A_{i,j} = A_{j,i}$ and therefore the matrix is redundant and symmetric. Figure 1B shows the adjacency matrix for our example.



|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 7 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

Figure 1: **A**: a simple undirected graph with 9 nodes and 8 edges that constitute two connected components. The shortest path between nodes 3 and 4 is highlighted in red. Note that there can be multiple paths between a pair of nodes. **B**: the adjacency matrix of the graph.

In real networks most nodes tend to be connected only to a small minority of all nodes, which results in 'sparse' adjacency matrices that are filled mostly with zeros. A simple way to compact the definition of such graphs (and speed up the handling of connections) is to give a listing of the edges (also called an *incidence list*) instead. For many purposes, this is best implemented by an $m \times 2$ matrix, in which each row defines a pair of nodes connected by an edge and $m$ is the number of edges. The graph in our example can thus also be defined as:

| | |
|---|---|
| 1 | 2 |
| 1 | 5 |
| 3 | 6 |
| 3 | 7 |
| 3 | 9 |
| 4 | 8 |
| 6 | 9 |
| 8 | 9 |

Instead of a $9 \times 9$ matrix[a] we ended up with a $8 \times 2$ matrix containing the same amount of information. Gains in storage can be even greater with larger sparse networks. On the other

[a]This matrix actually has only 36 informative elements - why is that?

hand, the generation of new links is sometimes easier using an adjacency matrix, and all links of a single node are immediately accessible as a row (or column) of an adjacency matrix, but need to be searched for in an incidence list. Which of the two structures is more efficient depends on the particular network we study and on what we intend to do with it.

Look at the script `netstart.r`. It contains two simple functions to generate two types of random networks. One is an Erdős-Rényi (ER) random graph, in which all pairs of nodes have the same probability to be connected by a link: this type of network is most easily generated by an adjacency matrix. The other example is a Barabási graph, which is built up by starting with a single node, then adding one node at a time. Whenever a new node is added, it is connected to some of the old nodes randomly, and the probability of the old nodes to receive one of the new links increases with their degree (i.e. with the number of links they already have). This results in those nodes with more links receiving even more, and as the graph grows, the distribution of node degrees approaches a *power-law* distribution of the form $P(k) \sim k^{-\gamma}$, where $P(k)$ is the probability that a node is connected to $k$ other nodes in the network and $\gamma$ is the exponent of the distribution. Such networks are also called *scale-free*[b] and they tend to have a long "tail" at high values of $k$, i.e. highly connected nodes[c] are more frequent than would be expected in simple random networks. Human contact networks tend to be scale-free. Look at the degree distribution of both types of networks implemented in the starting script. Generate large networks of both types with about the same number of nodes and links and compare their distributions. Try to fit a power-law function to the Barabási graphs you generate. Hint: power-law distributions are linear on a log-log scale; for linear regression between two vectors (`y` against `x`), you can use the function `lm(y∼x)`. You can view summary of the fit with `summary(lm(y∼x))` and you can plot the regression line to your log-log plot of the distribution with `abline(coef(lm(y∼x)))`.

The degree distribution of an Erdős-Rényi network follows a Poisson distribution. Can you guess the parameter of the distribution? Generate a large Erdős-Rényi graph (with, e.g., 1000 nodes), and try to find the Poisson distribution that fits to the degree distribution of this network. Use the `dpois` function to generate the density function of the appropriately parameterized Poisson distribution, and then plot it over the observed degree distribution of the actual Erdős-Rényi network. Finally, generate a large Barabási graph and an Erdős-Rényi graph with the same number of nodes and approximately the same number of edges. Compare the degree distributions of the two graphs. (If you want to generate even larger graphs, see exercise Eb2. below).

## 1.2 Linking epidemics to networks

To simulate epidemics over a network of contacts, we need to keep track of the infection status of all individuals. This is done most easily by defining a vector for the infection status, in which the `i`*th* element describes whether the `i`*th* invidividual/node of the network is infected (in the simplest case, with just a logical value `TRUE/FALSE`). For infectious diseases that induce immunity, the typical classification of infection status includes three possible states: *S=susceptible* (uninfected

[b]This expression is sometimes reserved for networks with $\gamma < 3$.

[c]The nodes with most connections are called *hubs* in scale-free networks. In the context of epidemics, such individuals can be *superspreaders*.

individuals with no immunity), *I=infectious* and *R=recovered* (individuals who are no longer infectious, but retain immunity to re-infection). This is exactly the same classification as is used in the classic continuous SIR model of epidemics[d]. However, simple SIR models only keep track of the total population sizes of each category, which implies that individuals are interchangeable and all individuals can transmit the disease to any other individual. In network models of epidemics, disease can only spread between individuals who are linked to each other in the network. The starting script includes the implementation of an epidemic over a simple network, with just uninfected and infected/infectious individuals. Run the model: experiment with different network and transmission parameters. Extend the model with immunity, i.e. implement a *recovered* state of the infection status.

## 2 Exercises

In general I suggest to work with your own implementation of the network(s), because that gives you complete freedom and flexibility. However, `igraph` tools can provide powerful help in the analysis and visualization of the graphs that you have created (all you need to do is convert your graphs into *igraph* type). You can also use the more complex network constructors of the package (e.g. `grg.game()`) and then convert the generated graphs into a format that you can handle in your simulations.

It is also possible to work entirely with igraph, in which case infection status and other attributes can be stored as vertex attributes. See the entry on *iterators* in the `igraph` manual. However, this is recommended for advanced programmers only: otherwise, it is very instructive to build your own implementation of the various networks.

Finally, note that the stochasticity of this modelling framework (the formation and dynamics of links and disease transmission involve stochastic processes) implies that each simulation run will be different. Therefore, to draw conclusions about your network, you will need to run multiple simulations with the same settings, and look at the distribution of outcomes (for which the `hist()` function can often be helpful).

### 2.1 Basic exercises

Eb1. Write functions that convert an adjacency matrix to an incidence list, and vice versa.

Eb2. The function that generates Erdős-Rényi graphs in the starting script has limitations on the size of the networks it can generate (the adjacency matrices get huge and will not fit into memory). Write a function that can construct large ER networks by generating an edge list.

Eb3. Write functions to generate the following types of networks. A simple Watts & Strogatz network, which is created by aligning the nodes to form a "circle", with each node connected to its two neighbours, and then adding a few random links that establish long-range

[d]You are advised to read the Introduction of the reader to the module 'SIR models of epidemics'.

connections. This is the simplest example for a “small-world” network. Optionally, you might generate networks in which the nodes are arranged at the grid points of a square lattice, with each node connected to its four neighbours and again some random long-range connections. Finally, write a function that generates a number of subpopulations, such that most of the links are within subpopulations and a few links establish connections between subpopulations. These networks provide examples for clustered graphs.

Eb4. In the epidemic model provided in the starting script, implement recovery of infected individuals to immune/recovered status (an *SIR model*). Run multiple simulations with fixed sets of parameters: observe the variability of the outcome. Plot a histogram of epidemic size (total number infecteds over the whole epidemic), and interpret the result. To better visualize the spread of the epidemic over the network, plot newly infected nodes and recovered nodes with two new colours, and plot the links that transmitted the infection in the last time step also in colour.

Eb5. Look at the effect of network structure on epidemic spreading. Compare network types by selecting parameters such that the average degree of the nodes would be the same in each network. In addition, test two types of reference with random mixing (no network structure). First, ignore links, select pairs of nodes that interact randomly in each time step. Second, run the continuous SIR model (see Level 1 module) with parameters matched according to the network model(s) that you are using for comparison. Is it possible to match the parameters between the continuous and the network model? Compare the initial spread of the infection, and the total number of infecteds (with recovery in all cases). Test also the effect of varying connectivity within each type. (Hint: the difference between network types might depend on the parameters, e.g. connectivity, transmission rate, etc.).

## 2.2 Advanced/additional exercises

Ea1. So far, we have worked with fixed networks. However, real contact networks are dynamic: links can break up, new links can form, individuals can leave and enter the network. Implement such processes in your network(s). Try to structure and parameterize the processes such that they create or maintain a given type of network (e.g. an ER or a Barabási graph). Look at the changes over time in network properties (e.g. degree distribution, average degree etc.).

Ea2. Implement a heterosexual network for the modelling of a sexually transmitted disease (STD). Note that such a network constitutes a *bipartite graph*, which has two classes of nodes (males and females in this case) that can form links only with nodes of the other class. Compare the spread of an infection over a heterosexual network and over a homogeneous network with only one kind of individual.

Ea3. Devise optimal strategies of “targeted intervention” for various network types. E.g. if you have capacity to treat or vaccinate only 10% of the population, how should you choose that 10%? (For simplicity, assume that you have full knowledge of the network structure). Hint: check out `betweenness()` and other centrality functions in `igraph`.

Ea4. Scale-free networks tend to have a densely connected "core", including and linking the nodes with the highest degrees (the *hubs* of the network). Generate Barabási networks and try to find this connected core. You can develop your own methods, and you can also experiment with the algorithms implemented in `igraph` for this purpose, e.g. `graph.coreness()`, `leading.eigenvector.community()` or `walktrap.community()`.

Ea5. So far, the only information you have stored about individuals was their infection status. However, in such an individual-based framework, you can easily expand this with further information, as is needed by your particular research problem. In this exercise, consider the evolution of virulence in a network model. *Virulence* is defined as the harm an infection/pathogen causes to its host, e.g. in terms of an increased rate of death. To model its evolution, you need to implement variability and heritability of this trait. How can you do that? Furthermore, you need to define its effect, e.g. infected hosts (nodes) might die at each time step with a probability that depends on the virulence of their infection. To track evolution in the long term, you need also to add new susceptible individuals to allow for prolonged epidemics. First, implement virulence as an independent trait and observe its evolution: where is it converging to? Then, implement a *trade-off*[e] between virulence and transmissibility of an infection, i.e. define one of these traits an a decreasing function of the other. Where is virulence converging to now? How does it depend on the trade-off you have specified? You can also experiment with further variants: e.g. virulence could influence both the strength or the longevity of immunity induced in the host, and could also influence the ability of the pathogen to infect hosts with a given strength of immunity.

[e]Trade-off is a central concept in evolutionary optimization: it implies that one characteristic can only be improved at the expense of another.