A quick tour of IONTW*

Winfried Just† and Ying Xin‡

March 10, 2015

In this module[1] we guide you through some of the capabilities of IONTW. Highlights include the types of networks supported, setting up various types of models of disease transmission, observing the resulting dynamics, and collecting statistics on the outcomes. Along the way, the module also reviews some basic notions of graph theory.

1 First tour stop: Networks

Before you can work through any of our modules, you need to install NetLogo, IONTW, and all supporting files in one directory on your computer. After you have done so, open the program by double-clicking on the file `IONTW.nlogo` in the directory where you installed the software.

The IONTW interface that will eventually appear should look like in Figure 1. It has a lof of controls for various parameter settings. We will cover their use later in this module. Right now let us focus one the big panel in the middle. This is called the **World** window and is initially black. When you click **New**, a picture of your model of the contact network will appear in this window.

Contact networks are modeled by *simple graphs.* From the mathematical point of view, a simple graph is a pair $G = (V(G), E(G))$, where V is a nonempty set of *nodes* or *vertices,* while $E(G)$ is a set of *edges,* that is, unordered pairs of distinct nodes. Thus each edge will be of the form $\{i, j\}$ with $i, j \in V(G)$ and $i \neq j$. IONTW represents nodes by little colored discs and edges by straight line segments that connect the discs.

When you click **New** in the default settings of IONTW you will see a lot of green disks that represent nodes, but no distinguishable edges. Let us change the parameter **num-nodes** to the right of the **World** window. Set

num-nodes: 20

In this and all subsequent modules we will only specify the parameter settings that the reader is supposed to change. The implicit assumption is that all other

†Department of Mathematics, Ohio University, Athens, OH 45701 E-mail: mathjust@gmail.com

‡Department of Mathematics, Ohio University, Athens, OH 45701 E-mail: yx123812@ohio.edu

[1]This version was originally posted at `https://qubeshub.org/iontw`

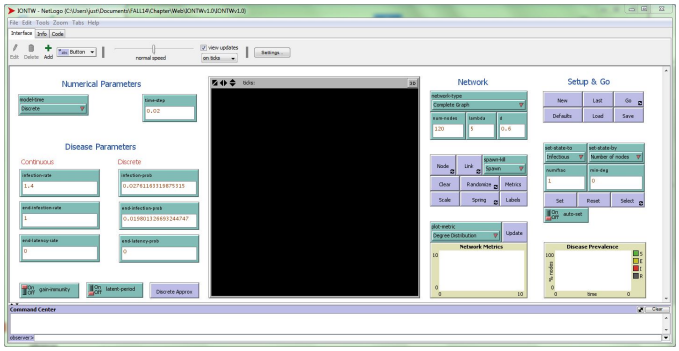

Figure 1: Opening screen of IONTW.

parameters are kept as in the previous step of the module. Therefore it is important that the user always works through the explorations and exercises of a given section of a given module exactly in the order in which they are written. If you need to interrupt your work, you can save a copy of IONTW with the current parameter settings by the option* File → Save As *in the upper left corner of the interface.

Click **New** again. This time a graph with clearly distinguishable edges will appear. Next see what happens if we change

num-nodes: 80

and then click **New**.

Exercise 1 *Why did we not see any distinguishable edges when we first clicked* ***New*** *right after opening IONTW? You may find out by clicking* ***Defaults*** *and noting the default setting of* ***num-nodes****.*

Now let us take a closer look at a very small network. Change

num-nodes: 4

and then click **New**. A nice graph with 6 edges will appear. As you can see, this graph contains all *possible* edges between its nodes. It is an example of a *complete graph.* Notice that the menu option **network-type** in IONTW's interface is set to **Complete Graph**, which is the default. If we want to find the representation of the graph in the **World** window as a mathematical object, we will need to see how the nodes are labeled. Press **Labels** to find out. Notice that the vertex set of this graph is $V(G) = \{0, 1, 2, 3\}$. While in mathematical explorations of networks and models of disease transmission on them it will usually be convenient to assume that the nodes are labeled $1, 2, \ldots, N$, the NetLogo software always assigns labels $0, 1, \ldots, N-1$. The complete graph with vertex set $V = \{0, 1, \ldots, N-1\}$ will be denoted by K_N.[2]

[2]In some of our mathematical explorations we may for convenience use the symbol K_N also for the complete graph with vertex set $\{1, 2, \ldots, N\}$.

Exercise 2 *Write out the edge set $E(K_4)$ of the graph K_4 in mathematical notation.*

IONTW gives you a lot of information about the properties of the network that is currently displayed in the **World** window. By default, the plot **Network Metrics** gives information about the **Degree Distribution** of the graph. The *degree* k_i of node i is the number of nodes j that are *adjacent* to i, that is, connected to i by an edge. The plot shows a histogram of the numbers of vertices with degree k for all relevant k. In K_4, the degree of each node is 3, and the histogram shows a single vertical bar of height 4. Note that the number 4 on the horizontal axis of this plot is obtained by adding 1 to the maximum degree; we will need to keep this feature in mind when we explore degree distributions of larger graphs.

Graphs in which each node i has the same degree $k_i = k$ are called *k-regular* or simply *regular*. Each complete graph K_N is a k-regular graph with $k = N - 1$.

Now let us see what other types of graphs IONTW allows us to explore. Click on **Complete Graph**. A long list of network options will appear. Most of them will be explored in detail in subsequent modules; here we will give you just a few highlights. Change

num-nodes: 12
network-type $\rightarrow$ **Empty Graph**

After clicking **New** you will see a graph with no edges whatsoever. Such graphs are called *empty graphs* as the set of edges is empty. An empty graph with N nodes is often denoted by $\bar{K}_N$, as the edge set is the complement of the complete graph with the same vertex set. All nodes in $\bar{K}_N$ are *isolated;* that is, each node i has degree $k_i = 0$. Note that the vertical bar in **Network Metrics** plot for the **Degree Distribution** ranges from 0 to 1 on the horizontal axis in this example.

Exercise 3 *Is it always true that $K_N \neq \bar{K}_N$?*

Now change

network-type $\rightarrow$ **Nearest-Neighbor 1**
d: 1

After clicking **New** you will see an example of a *one-dimensional nearest-neighbor* network: Each node is connected with its 2 nearest neighbors. Next change

d: 2, 3

When we write like this in our modules, we want you to first change **d** to 2, create a network and explore it, then change **d** to 3, create a **New** network and explore it.

What effect does the change of the parameter **d** have? A one-dimensional nearest-neighbor network with parameter **d** $= d$ and **num-nodes**$= N$ will be denoted by $G^1_{NN}(N, d)$.

Exercise 4 *(a) How is the parameter* ***d*** *related to the degrees of the nodes in $G^1_{NN}(N, d)$? (b) Find a sufficient and necessary condition for $\{i, j\}$ to be an edge in $G^1_{NN}(N, d)$.*

Change

num-nodes: 60

After clicking **New** you will see a graph whose edge set is difficult to make out. However, inspection of the **Degree Distribution** in the **Network Metrics** plot will show you that the degree of each node is still 6 (as long as you retained the parameter setting $\mathbf{d} = 3$ as per our admonition above!). It is often a good idea to double-check the visual information in the **World** window against this plot.

Next change

network-type $\rightarrow$ **Nearest-Neighbor 2**
d: 1

and click **New**. You will see an example of a *two-dimensional nearest-neighbor* network $G^2_{NN}(N, 1)$. In these networks, nodes are (usually) arranged in a rectangular grid.

This graph is no longer regular. It appears that the nodes in the corners have degree 2, the other nodes along the edges have degree 3, and the nodes in the middle have degree 4. Inspection of the **Degree Distribution** in the **Network Metrics** plot should confirm this observation.

Let us see what happens if we change

d: 3

After clicking **New** we get a lot of diagonal edges. It appears from inspecting the **World** window that the nodes in the corners have degree 5, but when we move the cursor along the horizontal axis of the **Degree Distribution** in the **Network Metrics** plot to the left edge of the lowest bar we find that the lowest degree in this network is 8.

Exercise 5 *How can you explain this discrepancy?*

Let us look at some examples of networks $G^2_{NN}(N, 3)$ for other choices of N. Visualize the networks for

num-nodes: 36, 33, 22, 23

and record the number of rows and columns in the resulting grids.

IONTW determines the numbers m of rows and n of columns by factoring $N = mn$ so that $m \leq n$ and m is as large as possible. In particular, if N is prime, we don't get much of a grid at all!

A *path* in a graph G from node i_0 to node i_ℓ is a sequence $P = (i_0, i_1, \ldots, i_\ell)$ of vertices such that $\{i_r, i_{r+1}\}$ is an edge in G for all $r < \ell$. The *length* ℓ of the path P is the number of edges in the path. The path P is *simple* if the vertices $i_0, i_1, \ldots, i_\ell$ are all distinct. Let us look at some examples. Change

num-nodes: 6
d: 1

Press **New** and look at the resulting network. $P_1 = (0, 2, 3, 5)$ is a simple path in this network; $P_2 = (0, 1, 3, 2, 4, 5, 3)$ is also a path, but not a simple one, and $P_3 = (0, 1, 2)$ is not a path as $\{1, 2\}$ is not an edge. The *distance* $d(i, j)$ between nodes i, j is the length of the

shortest path from node i to node j. For example, in the graph in your **World** window, we have $d(0,5) = 3$ and $d(1,2) = 2$. Any sequence (i) is considered a path of length 0, so that $d(i,i) = 0$ for any node i in any graph G.

Change

plot-metric → Shortest Paths

After clicking **Update** you will see the distribution of the distances in the **Network Metrics** plot. It ignores distances 0, and shows that there are 14 ordered pairs (i,j) with distance $d(i,j) = 1$. Nodes i,j will have distance $d(i,j) = 1$ if, and only if, $\{i,j\}$ forms an edge. For the graph $G^2_{NN}(6,1)$ in your **World** window this gives 14 ordered pairs (i,j) as each of the 7 edges $\{i,j\}$ corresponds to 2 ordered pairs (i,j) and (j,i). Moreover, there are 4 ordered pairs (i,j) of vertices in opposite corners with $d(i,j) = 3$, and 12 ordered pairs (i,j) of vertices with $d(i,j) = 2$.

If no path from vertex i to vertex j exists in a given graph G, then we define $d(i,j) = \infty$. A graph such that $d(i,j) < \infty$ for all nodes i,j is called *connected;* the graph $G^2_{NN}(6,1)$ in your **World** window is an example.

Now consider the path $P_4 = (0,1,3,2,0)$ in $G^2_{NN}(6,1)$. It has length $\ell = 4$, and all vertices $i_0, i_1, \ldots, i_{\ell-1}$ are distinct, while $i_0 = i_\ell$. A path with this property is called a *cycle.* In this definition we require that $\ell \geq 3$. The path $P_5 = (0,1,0)$ would not be considered a cycle. Graphs without cycles are called *acyclic;* connected acyclic graphs are called *trees.*

Let us look at an example of a tree. Change

network-type → Regular Tree
lambda: 3
d: 2

Click **New**. You will see a rather ugly-looking graph. Move the speed control slider at the top from is default setting **normal speed** to the extreme right. Then press **Spring.** This will give a nicer shape to your graph after a while. When the graph has taken a nice shape, press **Spring** again and then **Scale** to make it better fit your **World** window. You will see a connected acyclic graph, that is, a tree.

The structure of trees is best understood if we single out one vertex and call it the *root.* IONTW always makes node 0 the root for network type **Regular Tree.** The set of nodes at a distance ℓ from the root is called the *ℓ-th level* of the tree and will be denoted here by L_ℓ. For the example in the **World** window we get

$$L_1 = \{1,2\}, \qquad L_2 = \{3,4,5,6\}, \qquad L_3 = \{7,8,9,10,11,12,13,14\}.$$

For $\ell > 3$ we have $L_\ell = \emptyset$ in this example. The largest ℓ for which $L_\ell \neq \emptyset$ is called the *height* of the tree. Note that the input parameter **lambda** controls the height in this network type. The nodes at the highest level all have degree 1 and are called *leaves.* For each node i that is not a leaf, the number of nodes at the next higher level that i is adjacent to is controlled by the input parameter **d**.

Now change

network-type $\rightarrow$ **Erdos-Renyi**
num-nodes: 10
lambda: 1.5

Click **New** a few times and observe what happens. While the **network-type** options that we had explored so far always gave us a fixed network for the chosen parameter settings, the option **Erdos-Renyi** gives us *random networks* that may change every time we click **New**. The particular type of random graphs that are created by this option are called *Erdős-Rényi random graphs,* named after the two Hungarian mathematicians who first studied them in detail in [1]. The graphs that you see in your **World** window after clicking **New** are *instances* of Erdős-Rényi random graphs $G_{ER}(N, \lambda)$ that are drawn from a certain probability distribution that is specified by the parameters $N = 10$ and $\lambda = 1.5$. Figure 2 shows an example of such a graph.

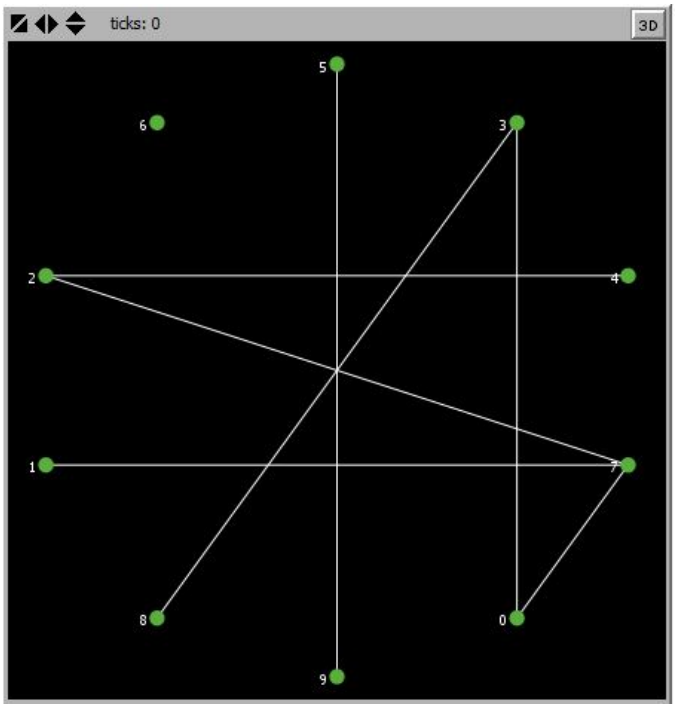

Figure 2: An instance of an ER random graph $G_{ER}(10, 1.5)$.

Notice that the graph in Figure 2 is not connected. For example, there is no path from node 6 to node 1. The graph has three *connected components,* where the connected component of a node i in a graph G is defined as the *subgraph* whose vertex set consists of all nodes j that are at a finite distance from i and whose edge set comprises all edges in the original graph G that connect these nodes. Connected components are examples of *induced subgraphs;* for details on this notion see [2]. Thus the connected components G_5, G_6 of nodes $i = 5, 6$ respectively of the graph in Figure 2 are:

$$G_5 = (\{5, 9\}, \{\{5, 9\}\}), \qquad G_6 = (\{6\}, \emptyset).$$

Exercise 6 *Find the connected component of node 1 in the graph G of Figure 2.*

You can visualize the connected components of a graph in the **World** window by using **Spring** and **Scale** as explained above.

The parameter λ in the construction of $G_{ER}(N, \lambda)$ represents the expected mean degree, denoted by $\langle k \rangle$, of an instance. For particular instances the mean degree $\langle k \rangle$ may be different from λ, but when we average over many instances, we should get a value very close to λ. Let us see how this works out.

Click **New** and then **Metrics**. Repeat about 8 times. Then click on the double-arrow icon on the **Command Center** bar. The window that pops up gives you a lot of statistics that IONTW collects when you click **Metrics**. The value of `Mean degree` gives you $\langle k \rangle$ for each instance. The actual values will be different from 1.5, but when you scroll down and average over all networks for which you collected data, you should get a value that is close to 1.5. We will discuss some of them later in this module and the remaining ones in later modules. Right now, record the values for `Number of connected components` and `Diameter of the largest component` for the network that is currently shown in your **World** window. It should be described by the last data set in the **Command Center** window. Look at the graph in your **World** window and check whether the number of connected components matches the data that IONTW reported. The *diameter* of a connected graph is the maximum distance between any two nodes. Check whether the value that IONTW reported for `Diameter of the largest component` matches the one that you would derive from inspecting the largest connected component of the graph.

You can delete the entries in the **Command Center** window by clicking **Clear** and minimize it by clicking the double-arrow icon again.

Let us look at a larger example of an Erdős-Rényi random graph. Change

num-nodes: 50
lambda: 2

Click **New**. For the graph that you now see in the **World** window you can still find the number of connected components and the diameter of the largest one by visual inspection after altering its shape with **Spring** and **Scale** as described above. However, especially for the diameter of the largest connected component this would be a bit tedious. After clicking **Metrics** you can easily look these numbers up in the **Command Center**.

Let us observe another important feature of the graph in your **World** window: While the mean degree $\langle k \rangle$ should be close to 2, not every node i will have degree $k_i = 2$. Some nodes i will be isolated and will have degree $k_i = 0$, some other nodes i will have degrees $k_i > 2$. The **Degree Distribution** in the **Network Metrics** plot will show you how many nodes i with degree $k_i = k$ there are for each k. The highest bar will most likely occur for $k = 2$ or for $k = 1$. You may want to click **New** a few times and observe how the degree distribution changes from instance to instance.

If we want a graph where each node has degree *exactly* 2, we need to use a different option for **network-type**. Change

network-type → **Random Regular**

After clicking **New**, visual inspection of the graph that you see in your **World** window will indicate that *each* node in this network has degree 2, and the **Degree Distribution** in the **Network Metrics** plot will confirm this observation. This graph is otherwise random. It has been drawn from a distribution that gives instances $G_{Reg}(N, \lambda)$ with parameters $N = 50$ and $\lambda = 2$. You may convince yourself about the inherent randomness by clicking **New** a few times. The instances $G_{Reg}(N, \lambda)$ of this construction are λ-regular graphs; we refer to them as *random λ-regular graphs.*

The options **Small World 1, Small World 2, Preferential Attachment, Generic Scale-free, Spatially Clustered** for **network-type** give yet other types of random graphs; we will cover them in detail in later modules.

IONTW also allows you to import custom networks. To see how this works, click **Load** and open the file `sample-network-detailed.txt` that you downloaded together with the code for IONTW. A network will appear in the **World** window. In this case we specified all the details of the network, including some colors for the nodes whose meaning will be discussed in the next section.

You can also specify a degree distribution instead of the details of the network. Change

network-type → **Custom Distribution**
num-nodes: 80

Then click **Load** and open the file `distribution.txt`. This will not immediately create a network, but will show you in the **Network Metrics** plot a distribution of degrees that you *ideally* would want in your network. Commit this histogram to memory or make a screenshot of it for future reference.

Now click **New**. One of two things may happen: The **World** window stays black and you see an error message `Degree sequence not realizable as an undirected graph!` in your **Command Center**. This will sometimes happen due to technical reasons that will be explained in a later module. In this case, simply click **New** repeatedly until a network appears in your **World** window. The **Network Metrics** plot will now show you the actual degree distribution. It should be rather similar to the ideal one we specified in the file `distribution.txt`, but most likely there will be small discrepancies. By clicking **New** a few more times you can observe how the network and its degree distribution change from instance to instance. This option gives you yet another method for constructing random networks.

2 Second tour stop: Initial states

Open IONTW, click **Defaults**, move the speed control slider to the extreme right, and change

network-type → **Regular Tree**

lambda: 3
d: 2

Click **New** to create a network, press **Spring** and wait until it has taken a nice shape. Then press **Spring** again, followed by **Scale** to make the network better fit the **World** window. Finally, press **Labels** to see how the nodes are numbered.

We already looked at this network in the previous section. In our intended applications of IONTW, the graphs represent contact networks and each node represents one host in a population. Green nodes represent susceptible hosts.

Now click **Set**. This will make one host infectious, which is signified by a red disc. We now have a prototypical initial state that corresponds to introduction of a single infectious host into an otherwise susceptible population. Much of our work in subsequent modules, and of work in mathematical epidemiology in general, focuses on the spread of infections within the given population for this type of initial state. The initially infectious host is often called the *index case* or *patient 0.*

Note that all the edges that connect the index case with other hosts are red, while the remaining edges are white. A red edge indicates that an effective contact between the two hosts that are represented by its endpoints will be *successful* (from the point of view of the pathogens), which means that it will lead to a new infection. Effective contacts along white edges will not be successful in this sense.

Now press **Reset**, followed by **Set**. Repeat a few times and observe how the number of the index case and the red edges change. By default, IONTW chooses the index case randomly.

We often will want to have more control over the choice of the index case. In this example, we may want to focus our attention on situations where the host is *not* a leaf, that is, has degree at least 2. We can enforce this by changing

min-deg: 2

Press **Reset**, followed by **Set**. Repeat a few times and observe which nodes are chosen as index cases. The selection is still random, but leaves are no longer chosen.

If we want to ensure that a particular node i, node 0 for example, becomes the index case, we can proceed as follows. Change

set-state-by → Vector from input

Click **Reset**, then **Set**. In the dialogue box that appears enter `[0]` and click **OK**. Be sure to include the square brackets.

Let us return to the setting

set-state-by → Number of nodes

Click **Set** without clicking **Reset** first. Then click **Set** a couple more times. Now several nodes will be infectious. Note that the edges that connect two infectious nodes are also white. While pathogens will be transmitted between hosts that have an effective contact along such an edge at the current time, this will not lead to a new infection. Similarly to contacts between two susceptible hosts, no effective contact between two infectious hosts will be successful from the point of view of the pathogens.

Repeatedly clicking **Set** to get an initial state with more than one infectious host is tedious. Let us try out a better method by changing

num/frac: 5
min-deg: 0

Click **Reset** and then **Set** to see how this works.

Certain control measures, such as vaccination, can be modeled by setting the state of some hosts prior to an outbreak to "Removed." To see how this can be implemented, change

set-state-to → **Removed**

Click **Reset** and then **Set**. You should see a state with 5 removed and no infectious hosts. Removed hosts are represented by grey discs, and the edges that have at least one grey endpoint are also displayed in grey as they play no role in the spread of the disease. In order to introduce an index case into such a partially vaccinated population, let us return to the setting

set-state-to → **Infectious**
num/frac: 1

Click **Set** without first clicking **Reset**.

Some of the options for setting the initial state that we have explored so far can be automated by using the **auto-set** switch. Change

network-type → **Nearest-neighbor 2**
d: 1
auto-set: On

Click **New** a few times and observe what happens.

Exercise 7 *How would you change the settings so that you can create random initial states in which exactly two of the nodes in the middle of this network will be initially infectious while all other nodes are susceptible without using the* ***Set*** *button?*

3 Third tour stop: Models of disease transmission

Open IONTW and click **Defaults**. Move the speed control slider to the middle of the slower range; adjust for comfortable viewing as needed. Change the following parameter settings:

model-time → **Continuous**
num-nodes: 10
auto-set: On

Click **New** to create a network and an initial state with one infectious node. This will set up a continuous-time model of type SIR, as the default setting **On** for **gain-immunity** specifies that hosts will gain permanent immunity upon recovery and the default setting **Off** for **latent-period** specifies that there is no **E**-compartment. The default choice of **Complete Graph** for **network-type** gives a model that embodies the uniform mixing assumption.

We should expect that when we simulate an outbreak in this model, we will see some new infections (nodes turning red), and eventual removal of each infectious node, which will cause these nodes to turn grey. At the end of the outbreak we may or may not have some nodes that escaped infection and stayed susceptible, that is, green. Let us start a simulation by clicking **Go** and see what will happen. You can repeat the experiment by clicking **New** or **Last** and then **Go** again. The difference in this case is that **Last** will give you an initial state with the same index case as before, and **New** will give an initial state with a randomly chosen index case.

With high probability you will observe that all nodes eventually turn grey, that is, all hosts experience infection during the outbreak. To see why this should be so, press **Metrics** and then look up the value `R0` in the **Command Center**. The reported value for `R0` shows that in this model, the basic reproductive ratio can be estimated as $R_0 > 5$, which is very high. Let us change

infection-rate: 0.14

This will drastically decrease the rate β at which a given pair of adjacent hosts makes effective contact. Click **New**, then **Metrics**, and look up the value of R_0 for this new model in the **Command Center**. It should be slightly larger than 1. Run about 5 simulations by clicking **Go** and then **New**. Observe the effect of the change in the parameter β on the outcomes of the simulations.

Now let us add an **E**-compartment to our model. Change

end-latency rate: 1
latent-period: On

Click **New**, then **Metrics**, and convince yourself that the change in the model does not alter the estimate of R_0. The new model is of type $SEIR$ though, and susceptible (green) hosts will first enter the "Exposed" state before eventually becoming infectious (red). Think about this state as a traffic light that has changed from green to yellow before eventually switching to red. Click **Go** and observe what happens. Repeat a few times.

Now let us set

end-infection-rate: 0

Click **New**. In the new model, infectious hosts leave the **I**-compartment at a rate of $\alpha = 0$, which is a mathematician's way of saying "never." The model we have set up is of type SEI. When you try to compute **Metrics** for such a model, you will get an error message. The reason is that for SI- and SEI-models, R_0 is undefined. If you do get this message, you can return to the interface by first clicking **Dismiss** and then the tab **Interface**.

Now start a simulation with **Go** and observe what happens: Eventually, the whole population will be in the **I**-compartment and will never leave it. Take a look at the **Disease Prevalence** plot. It shows you by means of color-coded curves, how the percentages of nodes in the **S**-, **E**-, and **I**-compartments changed over time during the simulation.

So far we have explored only continuous-time models. Now let us explore some discrete-time analogues. Change

model-time → Discrete

Now we need parameters that are probabilities instead of rates. We can convert the rates automatically into corresponding probabilities by pressing **Discrete Approx**. IONTW will compute corresponding values of transition probabilities and put them into the fields **infection-prob**, **end-infection-prob**, and **end-latency-prob**. We should expect similar outcomes of the simulations for the discrete and continuous versions of our SEI model. Click **New**, then **Go**, and observe what happens.

All nodes will eventually turn red and stay red, but when observing the **Disease Prevalence** plot, you will see that in the discrete version IONTW no longer terminates the simulation automatically while there are still infectious host. You will need to terminate it manually by pressing **Go** again.

Let us set up and run a discrete-time analogue of the $SEIR$-model that we explored previously. Change

end-infection-rate: 1

and press **Discrete Approx**. For these parameter choices we should get a model with a very similar value of R_0 as in the continuous-time version. You may want to confirm this by clicking **New**, then **Metrics**, and looking up the value of R_0 in the **Command Center**. If you enlarge it with the double-arrow icon you can easily compare the new value of `R0` with the one for the previous model.

For an almost identical value of R_0 we should expect similar outcomes of the simulations. You can try to confirm this prediction by simulating a few outbreaks. However, data on very few outbreaks are not very reliable and you might want to collect statistics on a large batch of simulations. In the next section we will introduce a tool for conducting such investigations.

One interesting class of discrete-time models are next-generation models. In these models we always set **end-infection-prob** to 1. We cannot obtain such models by using the automatic conversion **Discrete Approx** from continuous-time models and need to enter the probabilities manually. Let us set up a next-generation $SEIR$-model by changing:

time-step: 1
infection-prob: 0.15
end-infection-prob: 1
end-latency-prob: 0.1

The input parameter **time-step** controls how one step of the discrete model is related to NetLogo's internal time unit of a `tick` and may impact the quality of the display.

The new transition probabilities will give a model with a slightly larger R_0 than previously. Since **end-latency-prob** is small relative to **end-infection-prob**, we should expect that nodes stay yellow for much longer than they stay red. Click **New** and then **Go** and observe how this prediction plays out.

Now change

infection-prob: 0.05
gain-immunity: Off

latent-period: Off
num-nodes: 50
num/frac: 10

Since **gain-immunity** is switched **Off**, hosts will become susceptible again upon cessation of infectiousness. We also have eliminated the **E**-compartment by switching **latent-period Off**. Thus the model we have set up is of type SIS. Press **New**, then **Go**, and observe what happens. Nodes will oscillate in irregular patterns between the susceptible and the infectious state, the **Disease Prevalence** plot will show fluctuations in the percentages of infectious and susceptible nodes. Most likely, there will be no natural termination of the outbreak; you will need to stop the simulation by pressing **Go** again.

So far, we have only explored models where the contact network was chosen to be a complete graph. As our final example in this section, let us set up a next-generation $SEIS$-model on a two-dimensional nearest-neighbor network by changing the following parameter settings:

infection-prob: 0.7
end-latency-prob: 0.5
latent-period: On
network-type $\rightarrow$ **Nearest-neighbor 2**
num-nodes: 100
d: 1
num/frac: 3

Click **New**, then **Go**, sit back, relax, and enjoy the movie!

4 Fourth tour stop: Batch processing

Open IONTW and click **Defaults**. Change the following parameter settings:

model-time $\rightarrow$ **Continuous**
num-nodes: 10
auto-set: On

This will set up the SIR-model that we had already explored at the beginning of the previous section. There we found by running a few simulations that during a simulated outbreak the whole population tends to experience infection, with each node eventually becoming grey. Will this *always* be the case? If not, what is the probability that some nodes will escape infection? If all nodes will eventually become removed, how long will this take on average?

You can explore these questions with the methods that we covered in Section 3 by running several simulations and collecting statistics. The duration of a given simulated outbreak, in NetLogo's internal time units of `ticks`, can be found after termination of each run on the horizontal axis of **Disease Prevalence** plot. You may want to simulate a few outbreaks with clicking **New** and then **Go**. You will observe significant variability in the durations of simulated outbreaks.

It becomes clear that for reliable answers to the above questions we will need to collect statistics on a large number of outbreaks. Doing this by manually analyzing many individual runs would be extremely tedious. Fortunately, **NetLogo** has a built-in feature called "batch processing" that will allow you to collect such statistics relatively easily. To use this feature, open

Tools → BehaviorSpace

A dialogue box with the title **BehaviorSpace** will appear. It allows you to either define a **New** experiment or **Edit** an existing one. Right now click on **New** in the dialogue box. Another dialogue box with the title **Experiment** will appear. In this dialogue box you need to specify the parameters of your experiment. First enter a nice suggestive **Experiment name**. Make sure to choose a different name for each of your batch processing experiments so as not to lose previously saved output files from prior experiments. In the box titled **Vary variables as follows** the disease transmission and network parameters for a **New** experiment are set to the ones that currently appear in the interface. You can edit them, but right now we want to work with the ones that have already been specified in the interface.

In the box **Repetitions** you want to enter the desired number of simulation runs that you want to analyze. Let us enter 500.

The box **Measure runs using these reporters** allows you to specify the kind of data that you want to analyze in this batch processing experiment. Here we are interested in the number of hosts that are in the removed state and the time at the end of each simulated outbreak. Enter:

```
count turtles with [removed?]
ticks
```

Note that NetLogo refers to the agents in each simulation (hosts in our case) as `turtles` and uses a time unit called `tick`.

Since we are only interested in the values of these output variables at the end of each simulated outbreak, we want to *uncheck* the box **Measure runs at every step**.

The entries in the box **Setup commands** tell **NetLogo** how we want to initialize each simulation. Here we can simply enter

```
New-network
```

and the **auto-set** switch that we had previously set to **On** will take care of the initial state.

The preset choice `go` in the box **Go commands** will work fine for our purposes. Similarly, since we want to explore outbreaks in an SIR-model that are guaranteed to terminate on their own, the default setting `0` (no time limit) in the **Time limit** box will work here.

Now close the dialogue box by clicking **OK** and then click **Run**. If you inadvertently introduced a syntax error, NetLogo will complain and you need to fix it. To do so, **Edit** your experiment before clicking **Run** again. If the syntax is fine, a third dialogue box titled **Run options** will appear. Give instructions for how and where to save the output. Use the option **Table** that gives cleaner output. Leave the number of **Simultaneous runs in parallel** at its default value. Then click **OK**.

Next select the directory where you want to save your output. After you click **Save** the simulation starts running. A window should appear that has checkboxes **Update view** and **Update plots and monitors.** If the experiment runs slowly, you can speed it up by unchecking both and moving the speed slider in this window to the extreme right. When the window finally disappears, the experiment is completed.

As you can see, setting up and running a batch processing experiment involves a fixed sequence of steps during which you need to enter certain specifications. For your convenience, in the instructions *How to use these modules* at this web site we gave you a template for this sequence of steps; in the text of subsequent modules we will frequently refer to it. We strongly recommend that you keep all output files of your batch processing experiments in a dedicated directory in case you want to return to them later.

Now open your output file for the above experiment. It is a large spreadsheet that gives you information about all parameters of the simulations. Of particular interest are the last two columns that contain the data that we want to analyze. The column with the header `count turtles with [removed?]` gives you the numbers of hosts that experienced infection during each simulated outbreak. The last column gives you the durations of all simulated outbreaks, that is, the times when the last removals occurred.

Exercise 8 *Based on the data in your output file, find each of the following:*

(a) The proportion of outbreaks during which at least some hosts escaped infection.

(b) The proportion of outbreaks during which no secondary infections whatsoever occurred.

(c) The minimum, maximum, and mean durations of all outbreaks.

(d) The minimum, maximum, and mean durations of all those outbreaks during which all hosts experienced infection.

References

[1] P Erdős and A Rényi. On the evolution of random graphs. *Selected Papers of Alfréd Rényi, vol*, 2:482–525, 1976.

[2] Winfried Just, Hannah Callender, and M Drew LaMar. Disease transmission dynamics on networks: Network structure *vs.* disease dynamics. In Raina Robeva, editor, *Algebraic and Discrete Mathematical Methods for Modern Biology*. Academic Press, 2015.