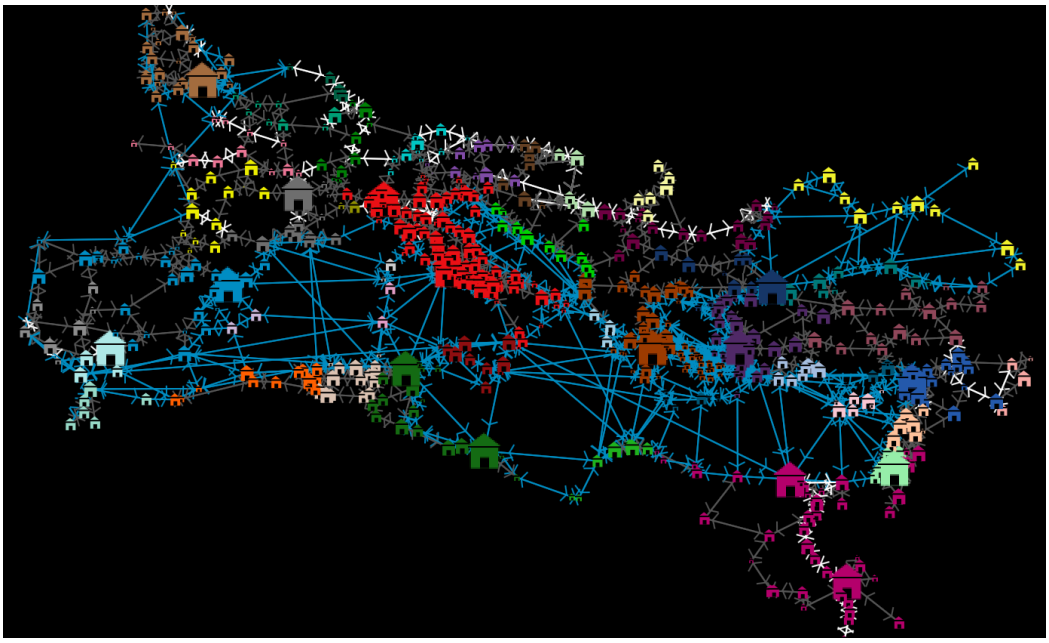
Importing a Roman Transport network with Netlogo

By Tom Brughmans
First version: Summer 2018
This version created 01/09/2018
Netlogo version used: 6.0.1
Extension used: nw (pre-packaged with Netlogo 6.0.1)
https://ccl.northwestern.edu/netlogo/6.0-BETA1/docs/nw.html
Cite this tutorial as:
Brughmans, T. (2018). Importing a Roman Transport network with Netlogo, Tutorial, https://archaeologicalnetworks.wordpress.com/resources/#transport .



1. Introduction

This tutorial provides an introduction to downloading open Roman datasets via https://projectMERCURY.eu and importing them into a Netlogo model. We will use the ORBIS dataset (http://orbis.stanford.edu/) to create a set of Roman settlements and major routes between them. We will analyse the network using the techniques in the nw Netlogo extension. This tutorial will reveal the importance of checking downloaded open data for mistakes and how you can correct them.

2. Conventions, tips and assumed knowledge

This tutorial assumes basic knowledge of Netlogo and of simulation. It is recommended to walk through the introductory tutorials on the Netlogo website or the tutorial on Netlogo for archaeologists on the Simulating Complexity blog:
https://simulatingcomplexity.files.wordpress.com/2014/07/dispersal_tutorial.pdf
https://ccl.northwestern.edu/netlogo/docs/
This tutorial will also refer to some network science jargon, concepts and techniques. You can get a basic definition of all of these from the glossary on my blog:
https://archaeologicalnetworks.wordpress.com/resources/#glossary
Good introductions to network science include the following:

Brandes, U., Robins, G., McCranie, A., & Wasserman, S. 2013. What is network science? Network Science 1(01): p.1–15.

Newman, M.E.J., 2010. Networks: an introduction, Oxford: Oxford University Press.

Code written in this tutorial will be formatted as in the following example:

```
breed [nodes node]
```

Save your project! Do this regularly and use multiple versions throughout the tutorial so that you can fall back on an earlier version at any time. I will remind you regularly throughout the tutorial to save your project.

If you are familiar with Netlogo: skip to section 6!

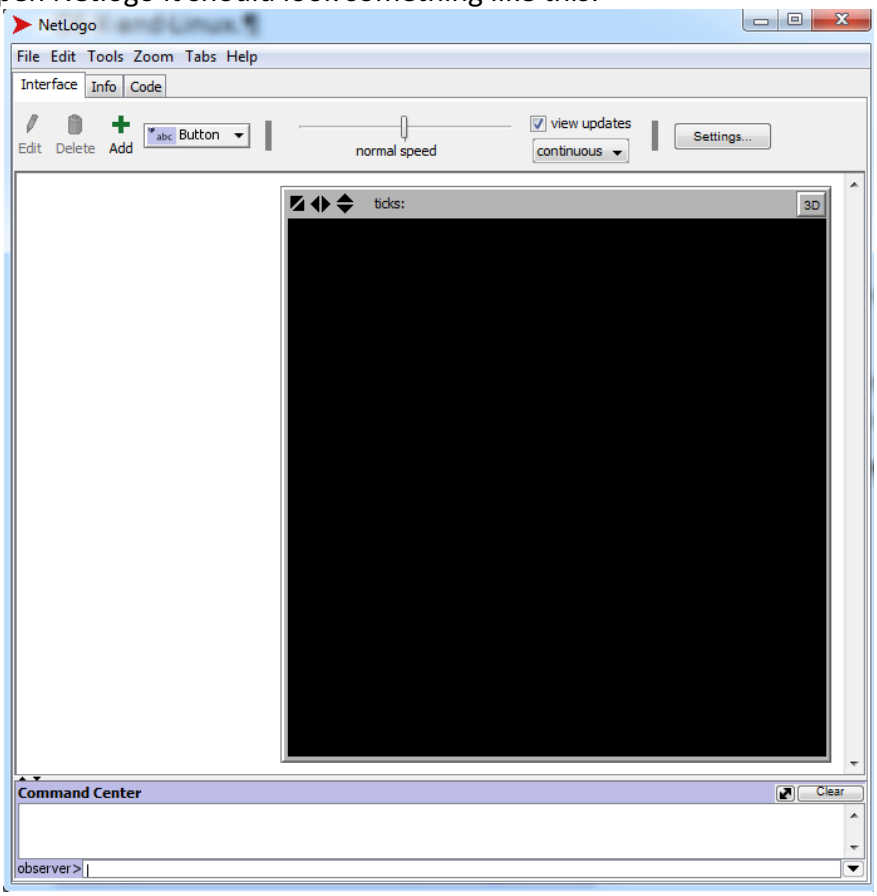
3. Download and install Netlogo

Netlogo is open source software and can be downloaded free of charge for Windows, Mac OS X and Linux.
Go to https://ccl.northwestern.edu/netlogo/download.shtml .
Download the Netlogo installer (this tutorial uses version 6.0.1).
Run the installer and install Netlogo.

4. Netlogo resources, manual and interface

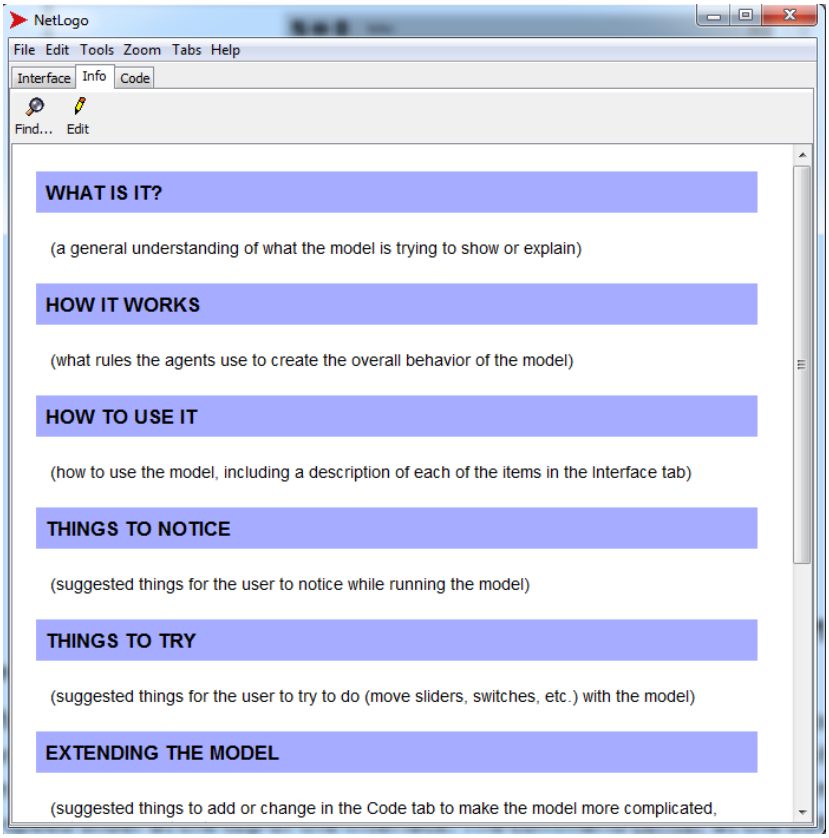
Netlogo has great documentation about all its features and code in its user manual:
https://ccl.northwestern.edu/netlogo/docs/
This manual includes tutorials, a reference to the software functions, a dictionary to its programming language, documentation of its extensions and much more.
Additional external resources can be found on the resources page:
https://ccl.northwestern.edu/netlogo/resources.shtml
A detailed reference to Netlogo's Interface can be found here:
https://ccl.northwestern.edu/netlogo/docs/
This tutorial will only give a very brief introduction to the key elements of the Netlogo interface you will be using throughout the tutorial.

When you open Netlogo it should look something like this:

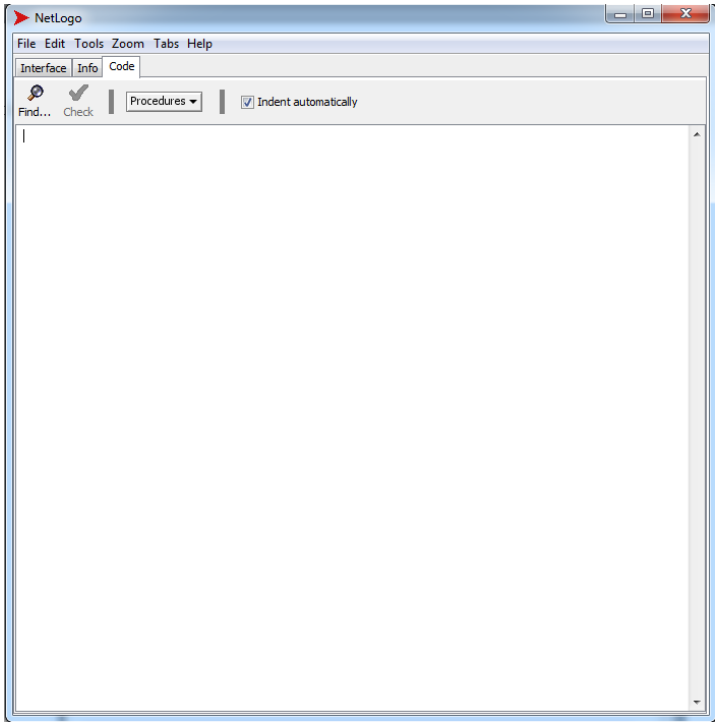


It has three tabs: Interface, Info, Code.

The **Interface tab** is where you watch your model run. Throughout the tutorial you will add buttons and sliders to control the variables of your model, and you will add monitors and plots to inspect what your model is doing. You can speed up or slow down the simulation using the speed slider at the top of the interface. The command center at the bottom will display messages you ask the model to produce, and you can also use it to give to commands to the model from the interface tab.



The **Info tab** is where you describe your model using a standardized set of questions. Adding this information when you share your model with other is crucial to enable them to work with your model. We will not work with the Info tab in this tutorial.

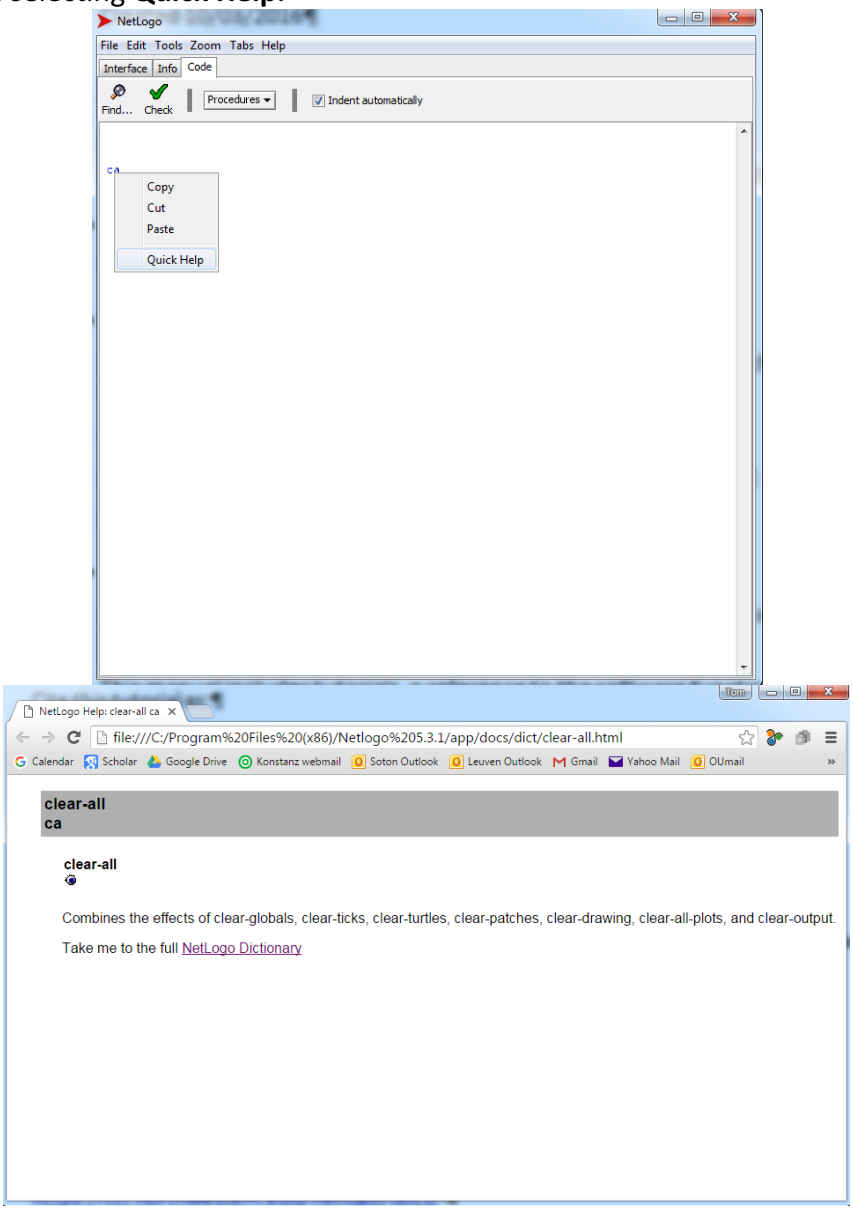


The **Code tab** is where you write and store the code for the model. In this tutorial we will be mainly working in the Code tab. A useful feature is the Check function at the top of this tab: click this to let Netlogo check your code for errors. If it finds errors then you will be guided to the error and asked to resolve it before you can continue, if it does not find errors then you can proceed with coding or viewing your model in action. NOTE: this error checker only

checks whether the primitives used and the order of the code comply with the Netlogo rules (i.e. the code's vocabulary and grammar). It will not check whether the code does what you want it to do, so getting no errors is no guarantee that the code works the way it should or the way you think it does. The error checker will always need to be used alongside other error checking techniques, like reporting variable values and checking them against expectations, or testing submodels independently.

5. Netlogo dictionary

A crucial resource when coding in Netlogo is its dictionary:
https://ccl.northwestern.edu/netlogo/docs/
For this tutorial, you will find the section on 'Links' in the Netlogo dictionary particularly useful, as well as the documentation of the 'nw' extension.
You can get direct access (even offline) to the entry about a particular primitive by right-clicking it and selecting **Quick Help**.



6. Create a working folder

Make a folder on your computer called 'Roman-transport'. All data we will use and the model we will create will be saved in this folder. When importing data into Netlogo models, as we will do in this tutorial, it is crucial that all data used in the model is saved in the same location as the model itself.

7. Download ORBIS data

ORBIS is the Stanford Geospatial Network Model of the Roman World (http://orbis.stanford.edu). It is a coarse-grained representation of the major terrestrial and maritime routes connecting the major settlements across the Roman world. Check out the ORBIS website if you want to learn more about it, or read the following publications:

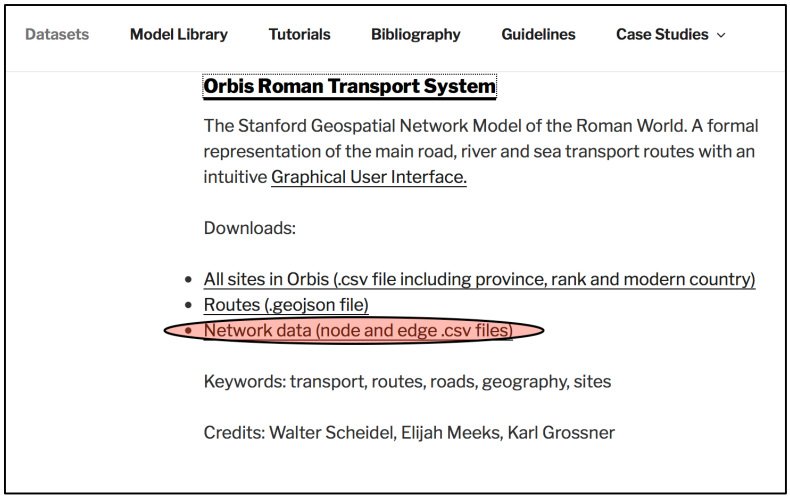
- http://orbis.stanford.edu/assets/Scheidel_64.pdf
- http://orbis.stanford.edu/assets/Scheidel_59.pdf

What is most crucial for this tutorial is that the ORBIS model can be reused with accreditation for research purposes. It offers a great starting point for studies of the Roman world that concern flows of goods, ideas and people.

Before we download this open dataset we need to think about how we want to use it in Netlogo. What we want to achieve in this tutorial is to have an agent-based model with a network representation of the Roman transport system where nodes represent settlements and edges represent routes. It will therefore be best to download the network data version of ORBIS.

Go to the MERCURY.eu page listing open Roman datasets:
https://projectmercury.eu/datasets/

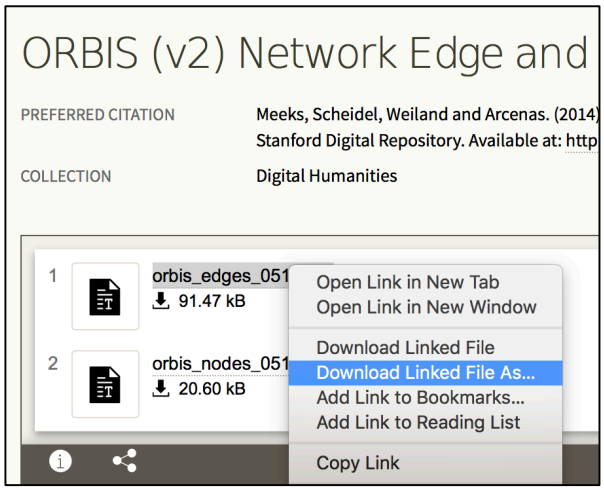
Scroll down the list and navigate to 'Transport System (ORBIS)'. You will notice from the description of this resource that we will need to download two datasets: a list of settlements/sites and a list of routes.



Click the link to download ORBIS Network data which will take you to a Stanford webpage:
https://purl.stanford.edu/mn425tz9757

You will see the license is specified as CC-BY-3.0 which means you can copy, redistribute, remix, transform, and build upon the material for any purpose: as long as you give appropriate credit.
https://creativecommons.org/licenses/by/3.0/

Download the two .csv files for the nodes and edges in this network (the nodes represent settlements and the edges represent routes). Right-click each file and select 'Download Linked File As …' and save them in the 'Roman-transport' folder:



A .csv file is a comma separated value file in which each line represents a data entry and each value in this entry is separated by a comma. It can be opened with a text editor or with a spreadsheet programme like Excel to view it as rows and columns.

The edges file with the route data has six data columns: the source node of the edge, the target node of the edge, the length of the route in km, the number of days to traverse the route, the expense of doing so, and the route type (road, coastal, overseas, …).

	A	B	C	D	E	F
1	source	target	km	days	expense	type
2	50001	50056	54.539	1.818	1.909	road
3	50001	50100	50.135	1.671	1.755	road
4	50001	50293	65.681	2.189	2.299	road
5	50001	50473	225.368	7.512	7.888	road
6	50002	50275	92.174	3.072	3.226	road
7	50003	50052	109.155	3.639	3.82	road

The nodes file with the settlement has four data columns: a unique node id, the settlement name as a label, the X and Y spatial coordinates.

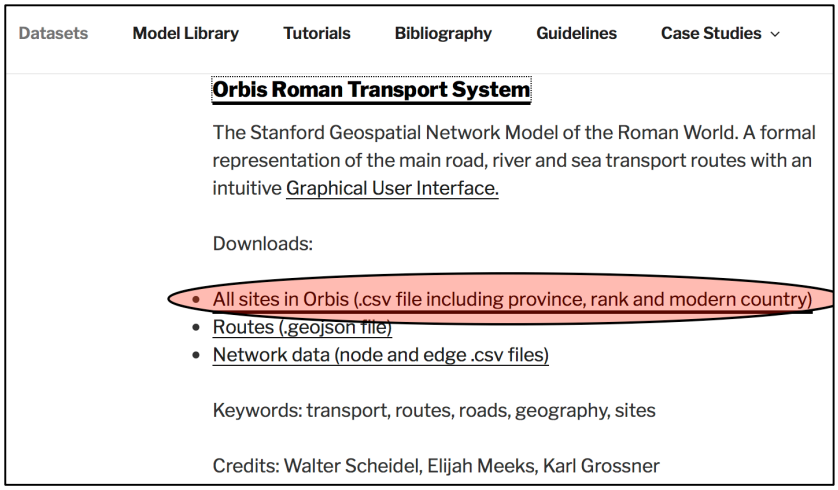
	A	B	C	D
1	id	label	x	y
2	50002	Ad fl. Tigrim	37.325	42.196
3	50209	Iulia Concorc	45.757	12.844
4	50327	Roma	41.892	12.486
5	50319	Praetorium	31.5	15.5
6	50294	Patavium	45.41	11.877

8. Errors in the data

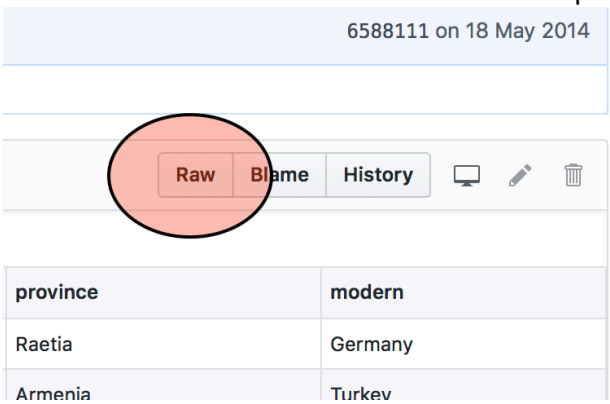
it's crucial to always thoroughly check downloaded open data to identify mistakes and omissions and to get an idea of how the data should be critically used in your study. Look

through the nodes file and try to find some of the settlements that have "x" as their label. Notice any mistakes? They don't have any coordinate information! In fact, there are a few issues with this nodes file. First, the X and Y coordinates are correct for some but switched around for most settlements. Second, the crossroads in the ORBIS model that are not towns do not have spatial coordinates associated with them. For these reasons it is advised to use another settlement dataset provided by the ORBIS team, which solves most issues with this file (but introduces others, wait and see…).

Go back to the Roman open datasets page of MERCURY.eu and now click 'All sites in Orbis':



This will take you to the Github page where this data is stored:
https://github.com/emeeks/orbis_v2/blob/master/sites_extended.csv
To download this file, click 'Raw', select all of the data (CTRL+A) and copy it (CTRL+C). Open a text processor such Notepad or TextEditor and paste the data into a new file (CTRL+V). Now save this new file as 'settlements.csv' in the 'Roman-transport' folder.



When you open this file in spreadsheet software you will notice it holds more information than the original nodes file: a unique settlement identifier (which will allow you to use this file as a replacement for the original nodes file), a label with the settlement name, a rank based on the Barrington Atlas ranking of settlements in the Roman World, spatial coordinates, cost and target columns which are empty, the Roman province the settlement was in and the modern country.

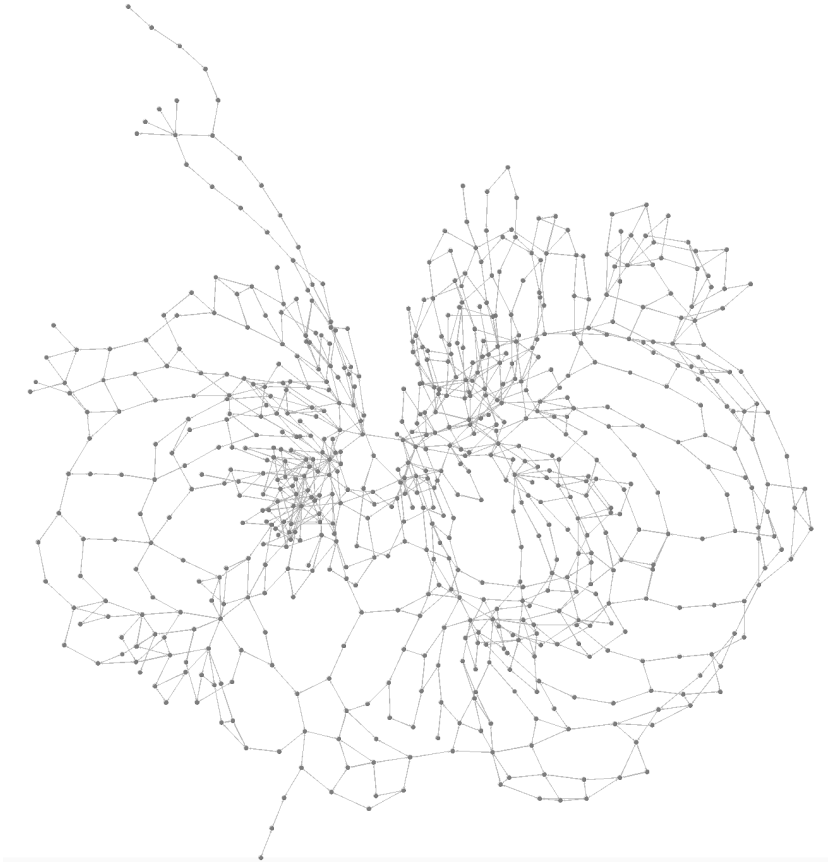
	A	B	C	D	E	F	G	H	I
1	id	label	rank	x	y	cost	target	province	modern
2	50001	Abodiacum	60	10.909	47.909	0	0	Raetia	Germany
3	50002	Ad fl. Tigrim	60	42.191	37.341	0	0	Armenia	Turkey
4	50003	Ad Publicanc	60	6.374	45.671	0	0	Narbonensis	France
5	50004	Ad Tricesimu	60	13.215	46.157	0	0	Italia	Italy
6	50005	Ad Aras	60	-0.977	38.736	0	0	Tarraconensi	Spain
7	50006	Adramyttium	80	26.937	39.504	0	0	Asia	Turkey

Using this settlements file instead solves the two issues mentioned above but it introduces another one: there are eight nodes with routes in the route file that are not included in the new settlements file. To solve this issue we could look them up in the original nodes file and add them to the spreadsheet. However, such a modification to the input data used is difficult to identify by colleagues who aim to use your model and reproduce its results. It would be much better if we could state explicitly in any publication we write about the model that the downloaded open access datasets are imported into the model as they are, and that all changes to this data is formally documented in the code of the model. In section 12 below we will do just that.

9. Prepare ORBIS data for Netlogo

Now that we have downloaded the ORBIS data, we need to explore how best to import it into Netlogo. The easiest way to import a network is to use Netlogo's 'nw' extension which has a feature for importing the network .graphml file format. Such a file can be very easily created in network software using the files you just downloaded. You could import the edges table into software such as Visone, and then attach the settlements.csv file as node attribute data. A crucial additional requirement of the 'nw' extension is to create in this .graphml file an edge attribute called 'BREED' and to set this value for each edge to 'routes'. Doing so will allow us to map these edges to a directed link breed called 'routes' (see below). Since describing this process in detail is outside the scope of the current tutorial, this has already been done for you (see this archaeological tutorial using the Visone software for a detailed explanation of how to create such a network file: https://archaeologicalnetworks.wordpress.com/resources/#Visone).

Go to the MERCURY.eu tutorials page where you downloaded this tutorial, and download the 'orbis.graphml' file. On this page https://projectmercury.eu/tutorials/ find the tutorial 'Importing a transport network' and click the download link under 'Input data'. The orbis.graphml file can be found in the downloaded folder, in a subfolder called 'data'. When you open this file in network software it would look like this:



10. Import Orbis data in Netlogo

Finally we can open Netlogo 6.0.1 for the first time in this tutorial!

Save a new model as 'Roman-transport.nlogo' in the 'Roman-transport' folder.

Write the following basic code to create a model that uses the 'nw' extension, a setup procedure and that is ready to include turtle/link/global variables:

```
extensions [nw]

turtles-own []

links-own []

globals []

to setup
  clear-all
  reset-ticks
end
```

The ORBIS network is a directed network, which means that an edge from A to B is a different one than the edge from B to A. In the case of ORBIS this difference was introduced by its creators because of the differences in sailing upstream or downstream rivers, or in the overseas routes taken. To ensure this feature of the dataset is correctly imported, we need to create a directed set of links. This can be done by creating a new link breed called 'routes'. Change everything above the setup procedure in your code to this, adding a new breed and the 'routes-own' variable placeholder:

```
extensions [nw]
```

```
directed-link-breed [routes route]

routes-own []

turtles-own []

links-own []

globals []
```

In addition to the nodes and edges, we also want to import all attribute information attached to the nodes and edges: settlement coordinates, names, provinces, etc. To ensure the 'nw' extension imports this information correctly, we have to define these attributes as in the routes-own and turtles-own attribute lists in the model:

```
extensions [nw]

directed-link-breed [routes route]

routes-own [ days expense km route-type ]

turtles-own [ node-id x y modern province rank ]

links-own []

globals []
```

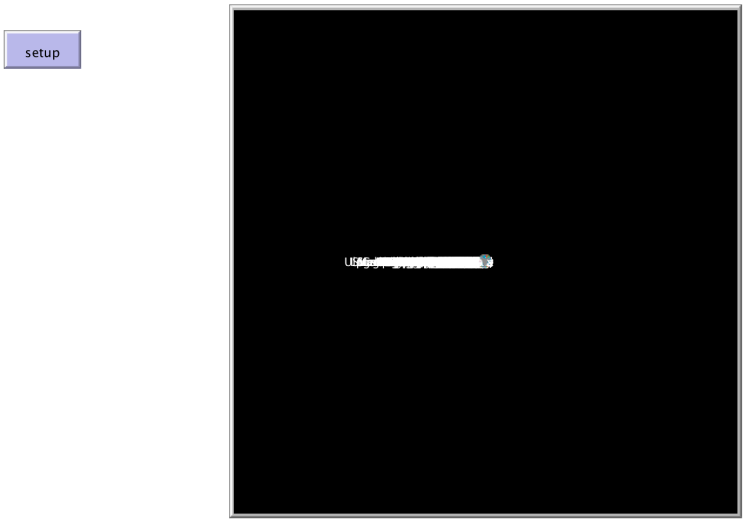
Subsequently we specify what sets of nodes and edges we will use with the 'nw' extension (setting the context in this way is a requirement of the 'nw' extension), and we upload the 'orbis.graphml' network file using the 'nw' extension. Change your setup procedure to this:

```
to setup
  clear-all
  nw:set-context turtles routes
  nw:load-graphml "orbis.graphml"
  reset-ticks
end
```

Note that because 'orbis.graphml' is saved in the same 'Roman-transport' folder as the model, we can simply refer to the file in the code and Netlogo will know where to look for the file. If it were note in the same folder then we would have to specify a path to the file.

We can now go to the interface tab and add a 'setup' button: click on the 'interface' tab, right-click the white space, select button, click in the white space again, in commands write 'setup', click ok.

Click the setup button to see whether it correctly loads the network. The correct result might surprise you: it looks like a comet-shaped white spot on the screen like this:



Save your model!

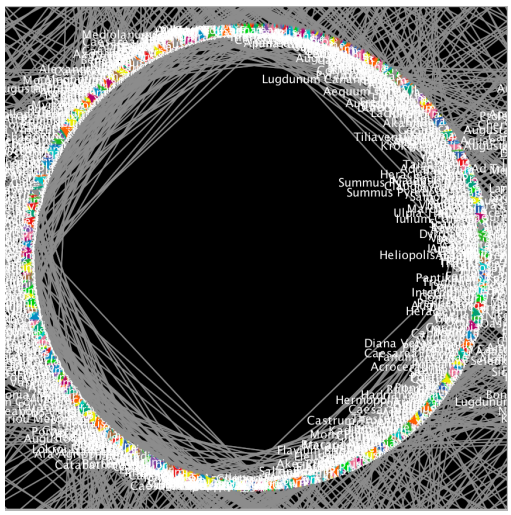
11. Position Roman settlements

This comet-shape is actually a good sign, it means the ORBIS network is correctly uploaded and as a result all nodes and their labels are plotted in the same place at the centre of the screen. Perhaps we can represent this network in a way that shows its structure a bit better by using layout algorithms.

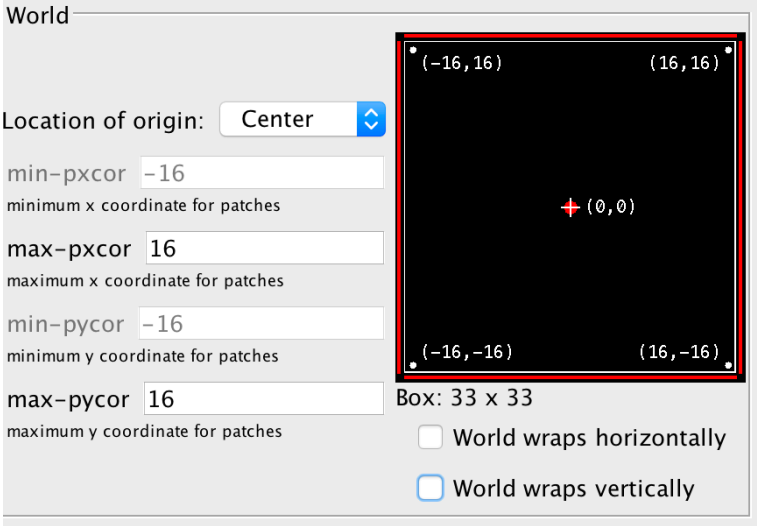
The simplest layout algorithm is to place all nodes in a circle using `layout-circle`. Change the setup procedure to include this:

```
to setup
  clear-all
  nw:set-context turtles routes
  nw:load-graphml "orbis.graphml"
  layout-circle turtles 15
  reset-ticks
end
```

When we now click the setup button we will see a circle, but all edges cross over the sides of the Netlogo world:



To stop edges crossing the sides of the Netlogo world we can change its properties by right-clicking the black space, clicking 'Edit…', and unticking the boxes for the world wrapping horizontally and vertically:



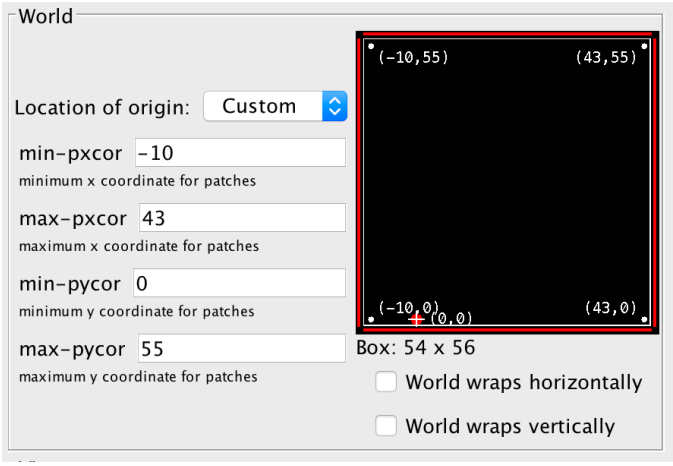
This circular layout gives a tidier representation of the network but it is not particularly insightful. Let's add a more appropriate geographical layout of the network by placing each settlement according to its x and y coordinates. Remember that the input ORBIS data has the variables x and y describing their geographical coordinates. We have given the turtles in Netlogo these variables. We will use this information to simply plot the settlements along the x and y axes of our Netlogo world. This is a simple and appropriate way of displaying this network if we do not use these locations in our analyses. If you wish to perform simulations that take the distance between Roman settlements in this Netlogo world into account, then it is better to use functions from the Netlogo GIS extension to position settlements.
https://ccl.northwestern.edu/netlogo/docs/gis.html
Look here for a good tutorial on the GIS extension:
https://simulatingcomplexity.wordpress.com/2014/08/20/turtles-in-space-integrating-gis-and-netlogo/

Go to the code tab, remove the circular layout line, and add code to the setup procedure setting the turtles' locations to the x y attributes:

```
to setup
  clear-all
  nw:set-context turtles routes
  nw:load-graphml "orbis.graphml"
  ask turtles
  [
    setxy x y
  ]
  reset-ticks
end
```

When you click the setup button now … you get a **Runtime Error**! The error says that the point where we want to position some agents is outside of the boundaries of the world. To solve this error we can modify the boundaries of our world to the extent needed to position all points. If you look into the settlements.csv input file you will notice that the lowest and highest values of the x variable are roughly -10 and 43, and for the y variable 23 and 55. We can use these values to modify the boundaries of our Netlogo world.

Right click the black space of the Netlogo world, click 'Edit', set the location of the origin to 'Edge' and set the coordinates as in this figure:

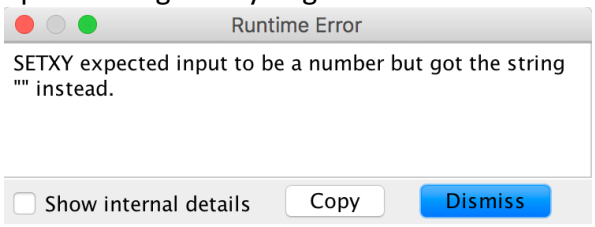


Note that Netlogo requires the origin point of 0,0 to be in the world so we have to set min-pycor to 0 rather than 23.

Save your model!

12. Formal documentation of data modification

When you press the setup button again … you get another **Runtime Error**!



This error indicates that there is an issue with some of the values of the x and y variables in the input data: they should all be numbers but some seem to be empty text strings.

This error revealed another issue with the input data, the issue we hinted at earlier in this tutorial: some of the settlements do not have x and y coordinates. In section 8 above we tried to correct some errors with the nodes dataset by using a different nodes input file: settlements.csv. However, the latter does not hold any information for 8 settlements which is why Netlogo cannot position some of the settlements.

Rather than correcting this mistake by modifying the input data, we could incorporate our modifications into the code of the model itself. This has a number of advantages:

- The open access version of the dataset used is imported without modification as input data in this model. There is no need for colleagues wishing to reproduce our results to download other data files.
- All modifications made to this input data are detailed formally in the code: they are unambiguously described.

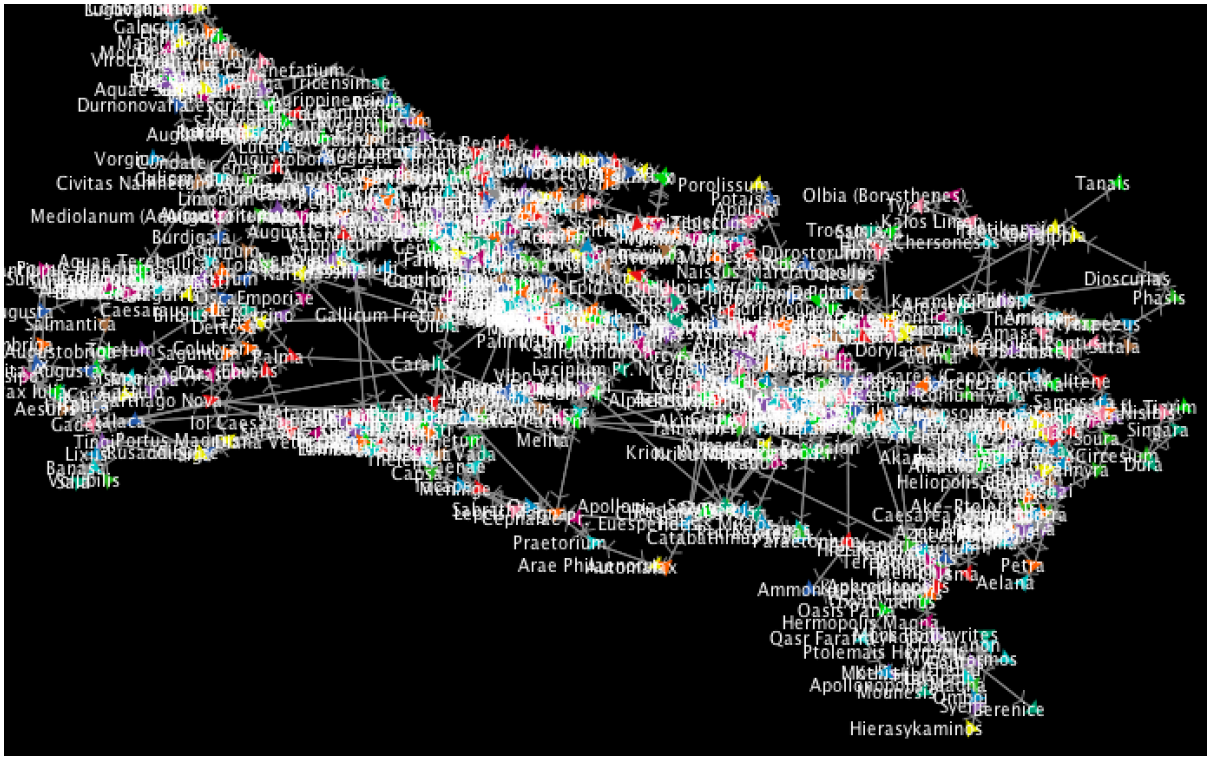
To correct this mistake in the model, we need to ensure we can correctly identify each node in the network. The input network orbis.graphml has a node attribute called 'node-id' which is the unique number that the ORBIS project uses to refer to each settlement. We imported this into Netlogo correctly by including it as a turtle attribute. We will use this node-id to identify the turtles that need correcting.

We could now add the corrections to the setup procedure, but this would result in a very long and messy setup procedure. Better to swing off these corrections into a procedure of their own.

Create a new data-correction procedure and include in it the lines below updating the x y values for eight turtles (just copy/paste the below into your model). Then call this data-correction procedure in the setup procedure just after you load the network:

```
to setup
  clear-all
  nw:set-context turtles routes
  nw:load-graphml "orbis.graphml"
  data-correction
  ask turtles
  [
    setxy x y
  ]
  reset-ticks
end

to data-correction
  ask turtles with [node-id = "50317"] [set x 12.258 set y 41.78 set label
  "Portus" set modern "Italy" set province "Italia" set rank 90]
  ask turtles with [node-id = "50522"] [set x 16.21 set y 41.36 set label
  "Aufidus" set modern "Italy" set province "Italia" set rank 60]
  ask turtles with [node-id = "50572"] [set x 25.213 set y 37.412 set label
  "Rheneia" set modern "Greece" set province "Graecia" set rank 60]
  ask turtles with [node-id = "50457"] [set x 23.589 set y 35.232 set label
  "Kriou Metopon" set modern "Greece" set province "Crete" set rank 60]
  ask turtles with [node-id = "50786"] [set x 25.75 set y 36.75 set label
  "Kerea" set modern "Greece" set province "Graecia" set rank 60]
  ask turtles with [node-id = "50789"] [set x 26.459 set y 37.005 set label
  "Lebinthos" set modern "Greece" set province "Graecia" set rank 60]
  ask turtles with [node-id = "50790"] [set x 23.625 set y 37.875 set label
  "Leros" set modern "Greece" set province "Graecia" set rank 60]
  ask turtles with [node-id = "50792"] [set x 25.37 set y 37.44 set label
  "Mykonos"set modern "Greece" set province "Graecia" set rank 60]
end
```

When you press the setup button again, you will see a rough outline of the Roman world, finally:



Save your model!

13. Visualise transport network

Now we have imported and corrected the ORBIS transport network, we can modify the way we visualise it in Netlogo to get get a better idea of the network structure. We will group the code of all these visual changes in a new "visualisation" procedure called in the go procedure just after the nodes are positioned in their XY coordinates:

```
to setup
  clear-all
  nw:set-context turtles routes
  nw:load-graphml "orbis.graphml"
  data-correction
  ask turtles
  [
    setxy x y
  ]
  visualise
  reset-ticks
end

to visualise
end
```

First of all, we clearly need to do something about those labels. We will store a turtle's label in a new turtle attribute `town-name` and we will add a line in the `visualise` procedure to remove all labels:

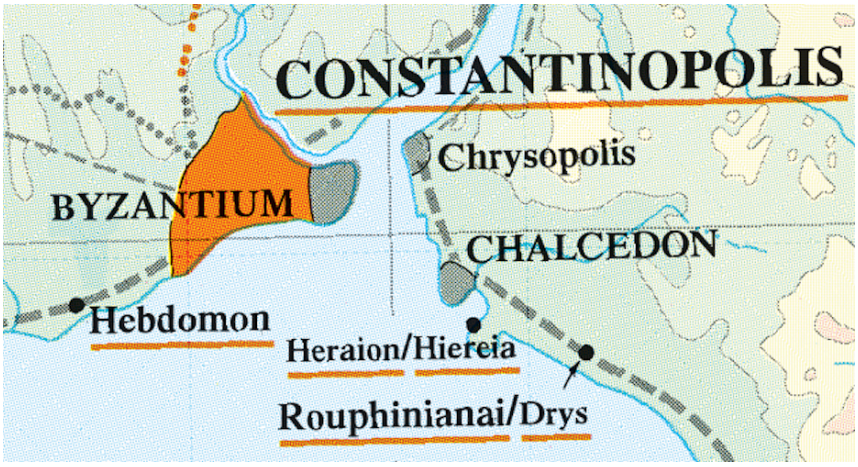
```
turtles-own [ node-id x y modern province rank town-name ]

to visualise
  ask turtles [set town-name label set label "" ]
end
```

If you now click the setup button the network will be much clearer.

Now we will change the way the turtles look. They represent settlements so a “house” symbol would be more appropriate:

```
to visualise
  ask turtles [set town-name label set label "" set shape "house"]
end
```

There are a lot of settlements in this ORBIS network, 678 to be precise. The parts of the Roman Empire with particularly dense distributions of settlements are very unclear, which we could solve by changing the size of turtles. Although we could just decrease the size of turtles overall, it would be good to use turtle size to represent something meaningful. The settlements have a “rank” associated with them, derived from their Barrington Atlas size: a generalizing qualitative indication of its importance. For example, in the following excerpt of map 52 Constantinopolis has a higher rank than Chalcedon, which has a higher rank than Hebdomon:



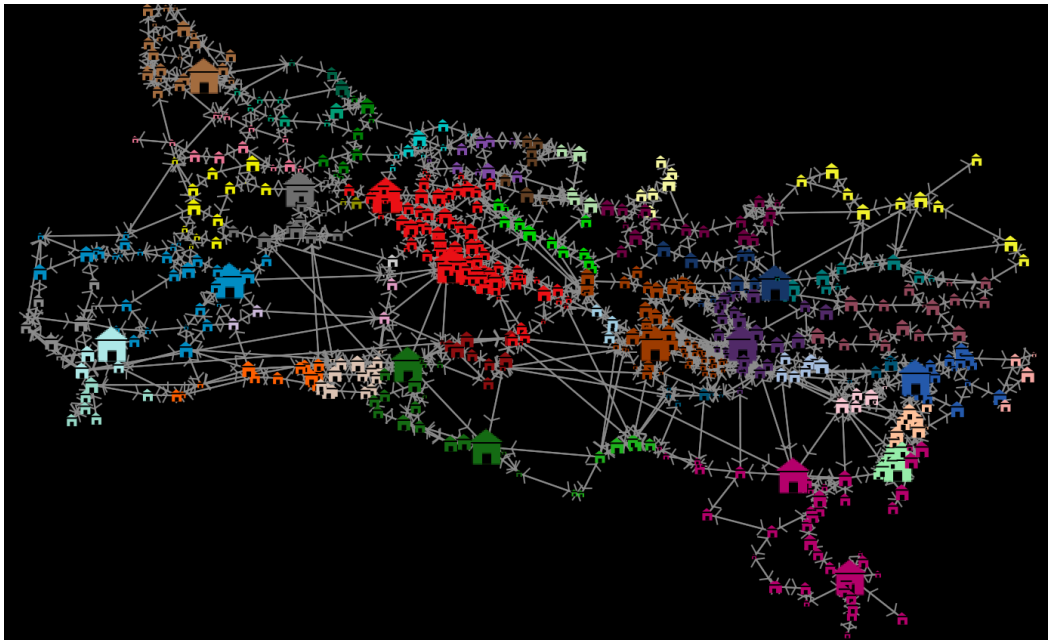
The ORBIS team converted this ranking into a 6 rank scheme with the following values: 6 (crossroads), 60, 70, 80, 90, 100 (the latter including settlements like Roma and Alexandria). Assign the following size values to each rank (copy/past the following):

```
to visualise
  ask turtles [set town-name label set label "" set shape "house"]
  ask turtles with [rank = 6] [set size 0.2]
  ask turtles with [rank = 60] [set size 0.3]
  ask turtles with [rank = 70] [set size 0.4]
  ask turtles with [rank = 80] [set size 0.7]
  ask turtles with [rank = 90] [set size 0.9]
  ask turtles with [rank = 100] [set size 2]
end
```

When you click the setup button the resulting network is still not particularly clear due to the many turtle colours. We can use the information about Roman provinces included in ORBIS to give all settlements in the same province the same colour. Create a new `provinces` procedure and create a button in the interface to trigger this procedure (copy/paste the following):

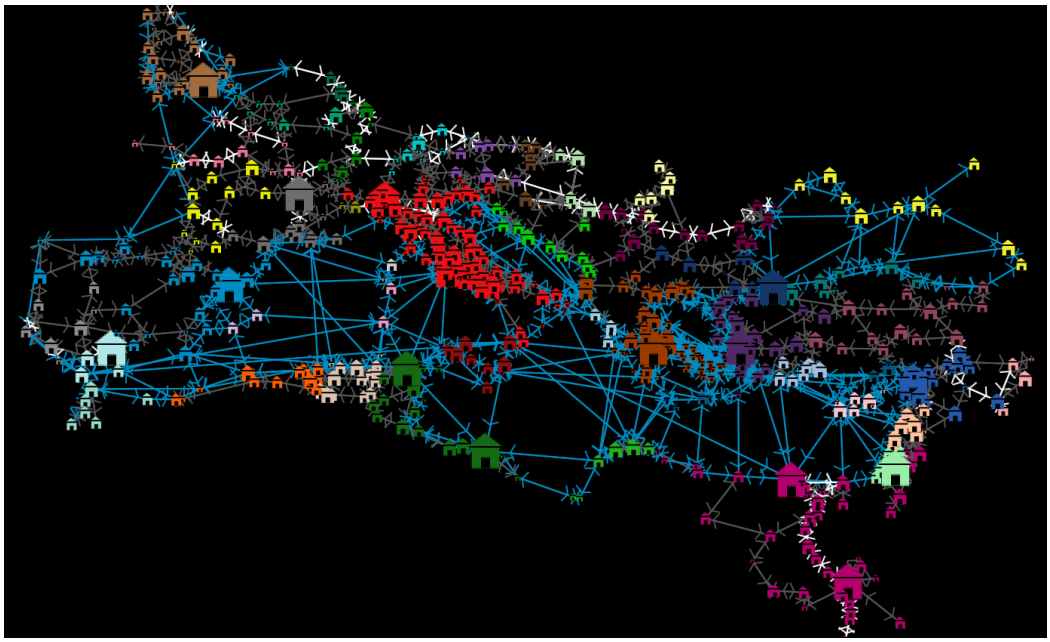
```
to provinces
  ask turtles with [province = "Italia"] [set color red]
  ask turtles with [province = "Lusitania"] [set color gray]
  ask turtles with [province = "Mauretania Caesariensis"] [set color orange]
  ask turtles with [province = "Britannia"] [set color brown]
  ask turtles with [province = "Aquitania"] [set color yellow]
  ask turtles with [province = "Cyrenica"] [set color green]
  ask turtles with [province = "Dalmatia"] [set color lime]
  ask turtles with [province = "Belgica"] [set color turquoise]
  ask turtles with [province = "Lugudunensis"] [set color pink]
  ask turtles with [province = "Raetia"] [set color cyan]
  ask turtles with [province = "Tarraconensis"] [set color sky]
  ask turtles with [province = "Syria"] [set color blue]
  ask turtles with [province = "Noricum"] [set color violet]
  ask turtles with [province = "Narbonensis"] [set color 4]
  ask turtles with [province = "Aegyptus"] [set color magenta]
  ask turtles with [province = "Arabia Petraea"] [set color magenta]
  ask turtles with [province = "Cilicia"] [set color 93]
  ask turtles with [province = "Bithynia"] [set color 83]
  ask turtles with [province = "Germania Inferior"] [set color 73]
  ask turtles with [province = "Germania Superior"] [set color 63]
  ask turtles with [province = "Africa"] [set color 53]
  ask turtles with [province = "Alpes Cottidae"] [set color 43]
  ask turtles with [province = "Pannonia Superior"] [set color 33]
  ask turtles with [province = "Moesia Superior"] [set color 123]
  ask turtles with [province = "Sicilia"] [set color 13]
  ask turtles with [province = "Macadonia"] [set color 23]
  ask turtles with [province = "Graecia"] [set color 23]
  ask turtles with [province = "Crete"] [set color 93]
  ask turtles with [province = "Thracia"] [set color 103]
  ask turtles with [province = "Asia"] [set color 113]
  ask turtles with [province = "Moesia Inferior"] [set color 123]
  ask turtles with [province = "Cappadocia"] [set color 133]
  ask turtles with [province = "Corsica"] [set color 8]
  ask turtles with [province = "Armenia"] [set color 18]
  ask turtles with [province = "Palestine"] [set color 28]
  ask turtles with [province = "Numidia"] [set color 38]
  ask turtles with [province = "Dacia"] [set color 48]
  ask turtles with [province = "Panonia Inferior"] [set color 58]
  ask turtles with [province = "Judea"] [set color 68]
  ask turtles with [province = "Mauretania Tingitana"] [set color 78]
  ask turtles with [province = "Baetica"] [set color 88]
  ask turtles with [province = "Epirus"] [set color 98]
  ask turtles with [province = "Lycia"] [set color 108]
  ask turtles with [province = "Baleares"] [set color 118]
  ask turtles with [province = "Sardinia"] [set color 128]
  ask turtles with [province = "Cyprus"] [set color 138]
  ask turtles with [province = "Outside_Blacksea"] [set color 46]
end
```

When you click the setup button again and then the provinces button the map looks much nicer and more understandable.



Now we will turn our attention to the routes. ORBIS recognises ten different route-types which are also imported as the routes-own attribute `route-type`: road, coastal, fastdown, fastup, overseas, downstream, upstream, slowcoast, ferry, slowover. We can colour-code the routes depending on their type. Let's make all maritime routes blue, all terrestrial routes dark grey and all river routes white. Add the following to the `visualisation` procedure:

```
  ask routes with [route-type = "road"] [set color 3]

  ask routes with [route-type = "coastal" or route-type = "overseas" or route-
type = "slowcoast" or route-type = "ferry" or route-type = "slowover"] [set
color sky]

  ask routes with [route-type = "fastdown" or route-type = "fastup" or route-
type = "downstream" or route-type = "upstream"] [set color white]
```



Save your model!

14. Analysing the network

In this section we will explore a few ways in which you can analyse the network you just imported into Netlogo. But why would you want to do this in the first place? There are a few reasons for this:

- Your simulation explores the flow of goods, information or people over a network (how fast could the message of the death of an emperor be spread to all towns in the Roman empire?).
- Your simulation concerns the evolution of a network structure (how did the infrastructure of the Roman road network change through time and what are the implications of these changes for the movement of people?).
- Your simulation draws on attribute information of settlements or roads (how was trade between places dependent on the theorised rank of settlements?).
- You want to use the settlements and routes as a visual background for your model.

In this section we will focus on examples of the first of these reasons, using network science methods included in the 'nw' extension of Netlogo.

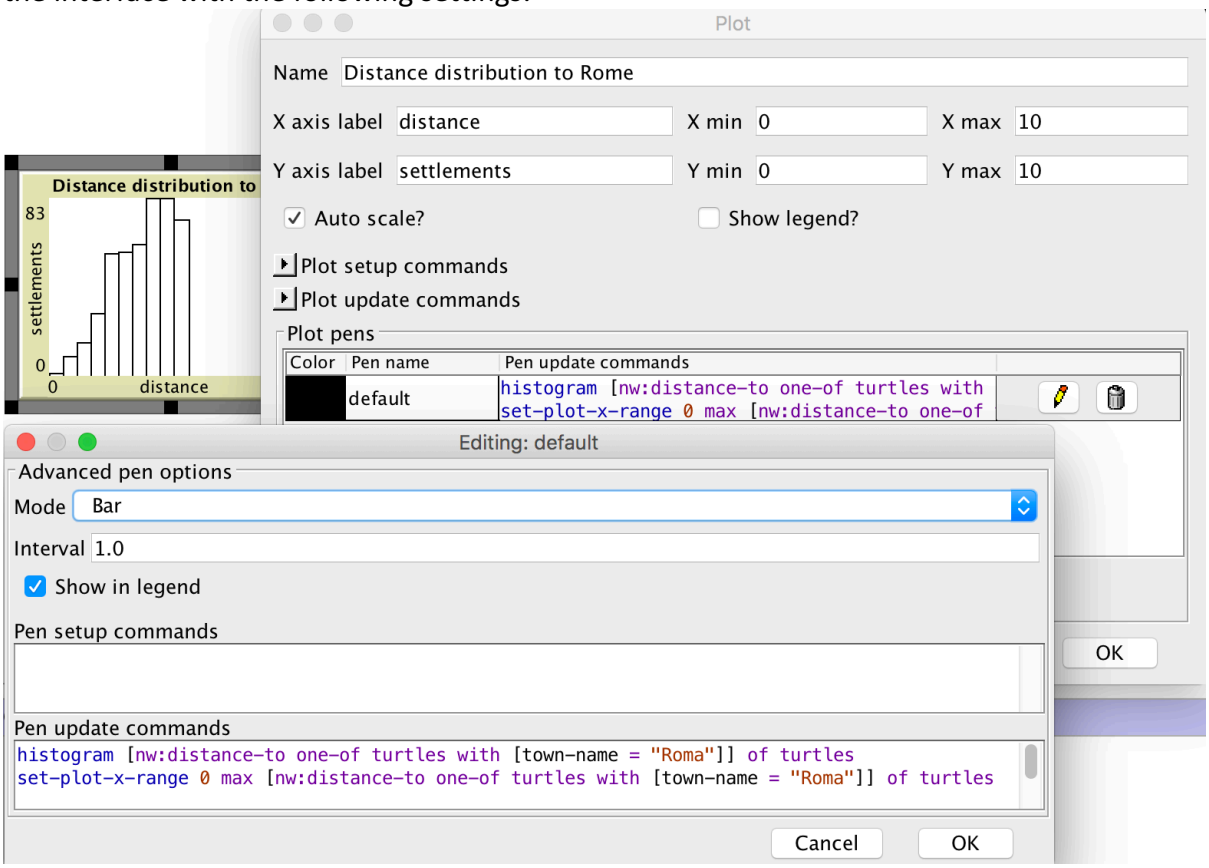
Let's explore the following question: **how fast could a messenger get to Rome from every town in the empire?** We can explore this by calculating the shortest path over the network from every settlement to Rome.

Create a new procedure called `analysis` below everything else in your code. For our first analysis we will use the command `nw:distance-to` to calculate the distance to Rome from each turtle. We will identify the closest settlement, the furthest settlement from Rome and the network distance from Alexandria to Rome (copy/past the following).

```
to analysis-distance
  type "closest settlement: "

  type [town-name] of one-of turtles with-min [nw:distance-to one-of turtles
  with [town-name = "Roma"]] type ", "

  print min [nw:distance-to one-of turtles with [town-name = "Roma"]] of
  turtles

  type "furthest settlement: "

  type [town-name] of one-of turtles with-max [nw:distance-to one-of turtles
  with [town-name = "Roma"]] type ", "

  print max [nw:distance-to one-of turtles with [town-name = "Roma"]] of
  turtles

  type "distance Alexandria: "

  ask one-of turtles with [town-name = "Alexandria"] [ print nw:distance-to
  one-of turtles with [town-name = "Roma"]]
end
```

Now create a new button in the interface to perform this analysis. When you click it, the command centre at the bottom of the interface will show you the results of the analyses. Note that the closest town to Rome in the network is … Rome itself.

You could also look at the distribution of these distances to Rome by creating a histogram in the interface with the following settings:



Note that the majority of settlements is at a distance of 8-10 steps removed from Rome. Perhaps an easier way to explore this information would be to modify the colour coding to represent these results. Create a new visualise-distance procedure, and create a button on the interface to trigger this procedure (copy/paste the below):

```
to visualise-distance
  ask routes [ set color black ]
  let maximum-result max [nw:distance-to one-of turtles with [town-name =
  "Roma"]] of turtles
  ask turtles
  [
    let my-result nw:distance-to one-of turtles with [town-name = "Roma"]
    set color (((( my-result / maximum-result ) * 10 ) * -1 ) + 9.9)
  ]
end
```

This code first calculates the maximum result, then it calculates the result per turtle, and then it normalises and inverses these results to a scale from 0 to 9.9 (because 9.9 is the code for the colour white in Netlogo) where 9.9 is represented by the colour white (closest to Rome) and 0 by black (furthest from Rome).

You can see that in general network distance to Rome increases with physical distance although coastal towns in general are closer than terrestrial towns.

Save you model!

But these results don't really represent how the world works: what matters for spreading a message is not so much the number of routes and towns you need to pass in the network, but rather the time or cost of doing so. This can be calculated by weighing the distance measure by the route attributes km (the distance of the route), days (the time it takes to traverse the route) and expense (the price to traverse this route).

We will create three new procedures to implement the analysis of network distance to Rome weighted by distance, time and expense. We will use the nw:weighted-distance-to command for this, and we will create buttons for each of these three procedures on the interface (copy/paste the below into your code and create three buttons):

```
to analysis-distance-km
  type "closest settlement: "
  type [town-name] of one-of turtles with-min [nw:weighted-distance-to one-of
turtles with [town-name = "Roma"] km] type ", "
  print min [nw:weighted-distance-to one-of turtles with [town-name = "Roma"]
km] of turtles
  type "furthest settlement: "
  type [town-name] of one-of turtles with-max [nw:weighted-distance-to one-of
turtles with [town-name = "Roma"] km] type ", "
  print max [nw:weighted-distance-to one-of turtles with [town-name = "Roma"]
km] of turtles
  type "distance Alexandria: "
  ask one-of turtles with [town-name = "Alexandria"] [ print nw:weighted-
distance-to one-of turtles with [town-name = "Roma"] km]
end

to analysis-distance-days
  type "closest settlement: "
  type [town-name] of one-of turtles with-min [nw:weighted-distance-to one-of
turtles with [town-name = "Roma"] days] type ", "
  print min [nw:weighted-distance-to one-of turtles with [town-name = "Roma"]
days] of turtles
  type "furthest settlement: "
  type [town-name] of one-of turtles with-max [nw:weighted-distance-to one-of
turtles with [town-name = "Roma"] days] type ", "
  print max [nw:weighted-distance-to one-of turtles with [town-name = "Roma"]
days] of turtles
  type "distance Alexandria: "
  ask one-of turtles with [town-name = "Alexandria"] [ print nw:weighted-
distance-to one-of turtles with [town-name = "Roma"] days]
end

to analysis-distance-expense
  type "closest settlement: "
  type [town-name] of one-of turtles with-min [nw:weighted-distance-to one-of
turtles with [town-name = "Roma"] expense] type ", "
  print min [nw:weighted-distance-to one-of turtles with [town-name = "Roma"]
expense] of turtles
  type "furthest settlement: "
  type [town-name] of one-of turtles with-max [nw:weighted-distance-to one-of
turtles with [town-name = "Roma"] expense] type ", "
  print max [nw:weighted-distance-to one-of turtles with [town-name = "Roma"]
expense] of turtles
  type "distance Alexandria: "
  ask one-of turtles with [town-name = "Alexandria"] [ print nw:weighted-
distance-to one-of turtles with [town-name = "Roma"] expense]
end
```

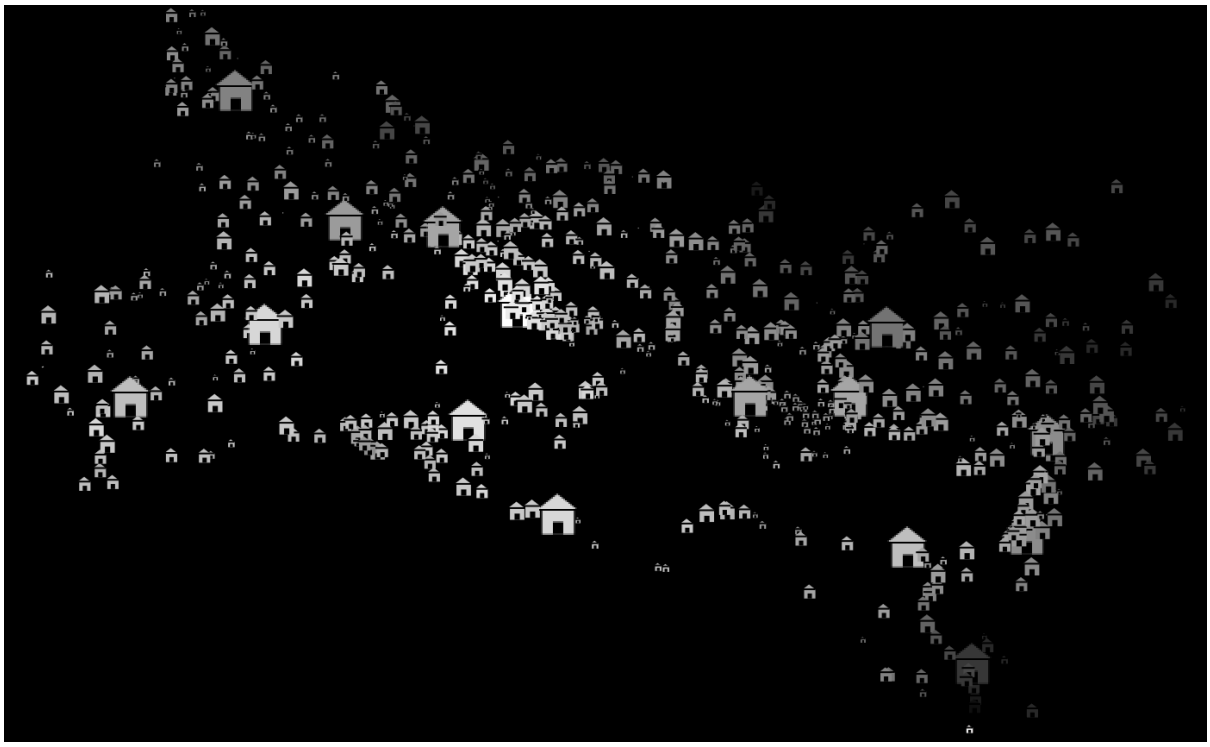
We will also create procedures to visualise these results in the interface, and add buttons for each of these three new visualisation procedures (copy/past the below in your code and create three buttons):

```
to visualise-distance-km
  ask routes [ set color black ]
  let maximum-result max [nw:weighted-distance-to one-of turtles with [town-
name = "Roma"] km] of turtles
  ask turtles
  [
    let my-result nw:weighted-distance-to one-of turtles with [town-name =
"Roma"] km
    set color (((( my-result / maximum-result ) * 10 ) * -1 ) + 9.9)
  ]
end

to visualise-distance-days
  ask routes [ set color black ]
  let maximum-result max [nw:weighted-distance-to one-of turtles with [town-
name = "Roma"] days] of turtles
  ask turtles
  [
    let my-result nw:weighted-distance-to one-of turtles with [town-name =
"Roma"] days
    set color (((( my-result / maximum-result ) * 10 ) * -1 ) + 9.9)
  ]
end

to visualise-distance-expense
  ask routes [ set color black ]
  let maximum-result max [nw:weighted-distance-to one-of turtles with [town-
name = "Roma"] expense] of turtles
  ask turtles
  [
    let my-result nw:weighted-distance-to one-of turtles with [town-name =
"Roma"] expense
    set color (((( my-result / maximum-result ) * 10 ) * -1 ) + 9.9)
  ]
end
```

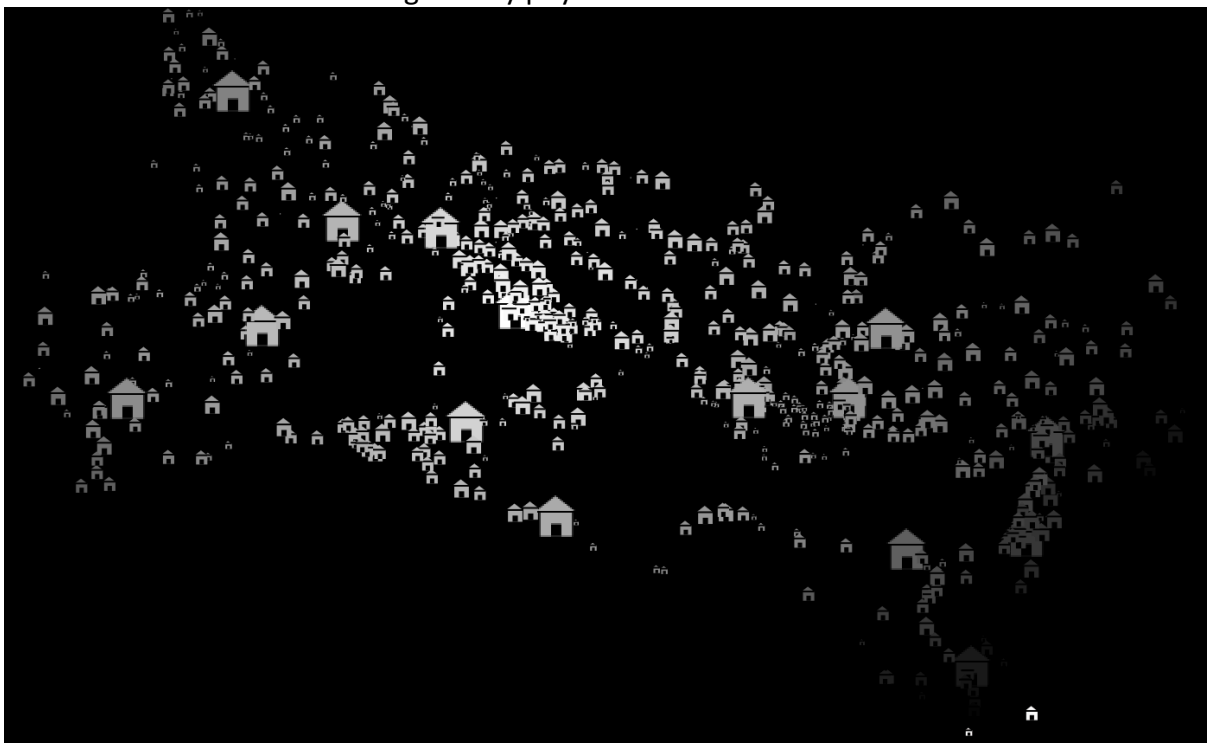
We have four different ways of exploring how far a messenger would need to travel from each settlement to the capital of Rome. The results are quite interesting and show interesting differences. They are best explored by dynamically clicking the four different boxes to identify the general differences. When considering physical distances, the eastern part of the Empire is very distant from Rome and the northern part much closer. When considering distance in days, the port towns are much closer than terrestrial towns, and this difference is much more pronounced when considering distance in expense (in ORBIS maritime transport is significantly cheaper per km that is terrestrial transport).

Save your model!

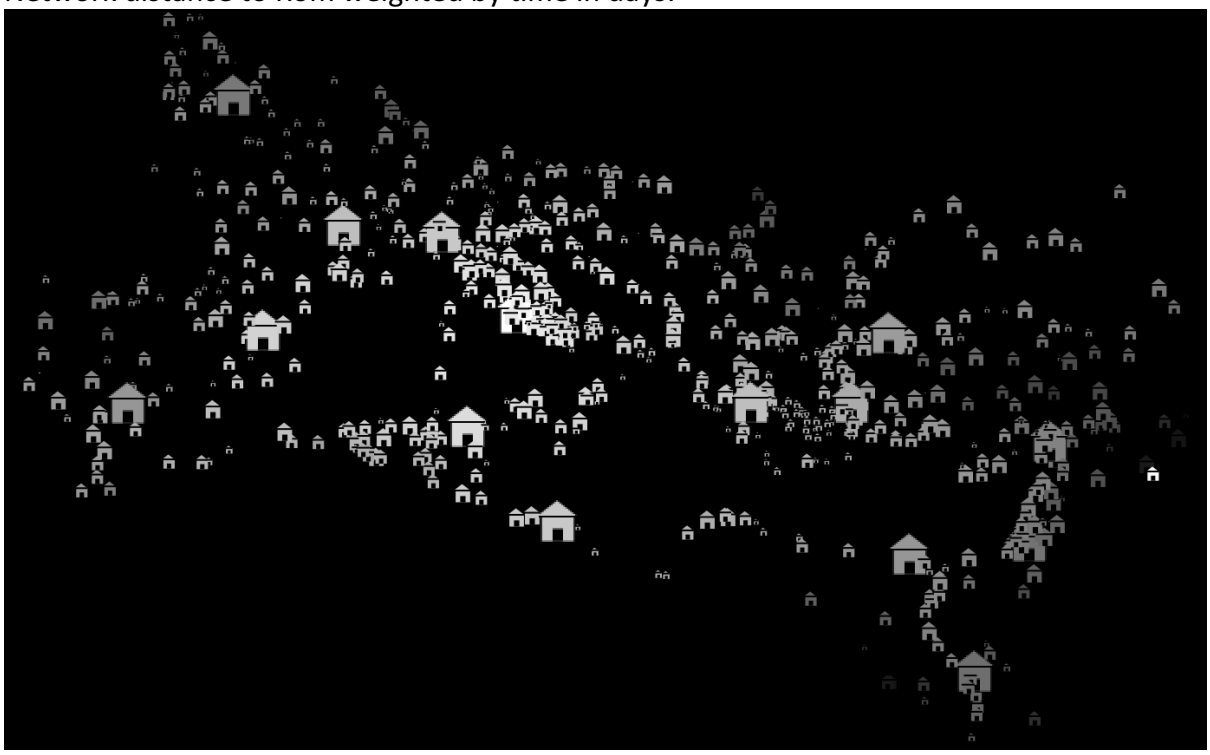
Network distance to Rome:



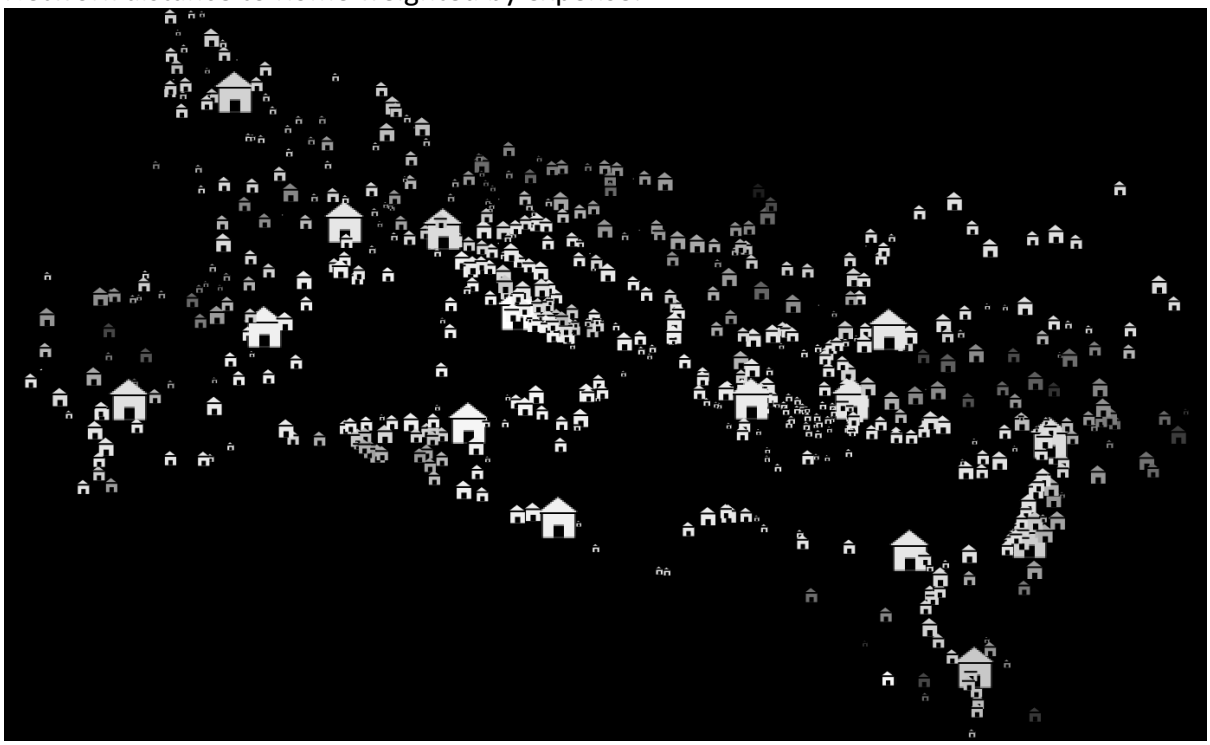
Network distance to Rome weighted by physical distance in km:



Network distance to Rom weighted by time in days:



Network distance to Rome weighted by expense:



15. Additional exercises

- Remove all towns in present-day Greece from the network and their associated routes using only code and not modifying the input data. What are the implications for the structure of the network and the flow of information through it?
- Use the "modern" node variable to colour code all nodes according to the present-day country they are in. Create two buttons in the interface that allow you to switch between colours of modern countries and Roman provinces.
- Establish how close on average Rome is to all other towns by using the `nw:closeness-centrality` command. Are there other towns that (in general) are more central and more close to all other towns than Rome?
  https://ccl.northwestern.edu/netlogo/6.0-BETA1/docs/nw.html