How to Visually Analyse Networks Using Gephi

AXEL BRUNS
A.BRUNS@QUT.EDU.AU
DIGITAL MEDIA RESEARCH CENTRE, QUEENSLAND UNIVERSITY OF TECHNOLOGY, AUSTRALIA

Network visualisations have become an increasingly common tool for many research fields. They may represent physical networks, visualising for example the flow of traffic on urban roads or the flow of electrical impulses through the brain; conceptual networks, such as the interconnections between companies in a sector of the economy or the relationships between components in a complex system; or communication networks, from patterns of correspondence between email accounts to interactions between users on major social media platforms such as Facebook or Twitter. In fact, in the field of communication studies the use of network analysis goes back at least as far as the groundbreaking studies of opinion leadership in small-town communities by Katz and Lazarsfeld (1955), well before the advent of modern digital media technologies – but the emergence of these digital "mass self-communication" tools (Castells, 2007) has increasingly shifted such interpersonal communication to digital platforms, and this in turn means that this form of communication is generating extensive digital trace data that can be visualised and studied with network analysis techniques.

This guide will introduce you to the basic concepts of network analysis and visualisation, and step you through the process of installing the leading open-source software for this work, Gephi, and of performing some basic network analysis and visualisation steps. Because network analysis can be applied to such a wide range of network data, and because the approaches to accessing such data are specific to each source, require in-depth considerations of data ethics, and are subject to constant change, it will not cover the procedures for accessing data themselves – we will work with the random network data that Gephi itself can generate, but you may substitute these data with any datasets you already have access to. The central aim of this guide is to provide you with the basic knowledge required for network analysis and visualisation, and to enable you to put that knowledge into practice using Gephi.

Why network analysis?

First, though, to some fundamental questions. For any given dataset, for instance of tweets contributing to a hashtag on the social media platform Twitter, we can already produce a range of quantitative, statistical observations: we can identify the accounts that are posting most actively; we can examine what other accounts are mentioned most frequently; we can analyse the texts of these posts and identify the most common keywords and phrases; we can use the posting times to chart the ebb and flow of conversations over time. Similarly, we can engage in detailed qualitative analysis: we can perform a close reading of the tweets (or, for larger datasets, at least of a representative sample of all tweets) to identify the key themes, tone, and sentiment of the discussion; or we can engage in a formal coding of tweet contents, using an appropriate codebook that we have developed for the case.

What, then, can network analysis contribute that such general quantitative and qualitative approaches cannot offer? The answer is that it enables us to consider the structure of interactions and interconnections between participants in our analysis. A basic statistical analysis of posting volume, for example, might identify some of the most active accounts – but a network analysis can tell us whether these accounts are well-integrated into the community of participants (whether they are opinion leaders, in the language of Katz and Lazarsfeld), or whether they are ignored by the rest of the community; in fact, the network analysis, or more accurately our interpretation of the structures revealed by the network analysis, can tell us whether there *is* a community of participants in the first place or whether they are all simply posting their views without actually engaging with each other.

It is important to note here that none of these approaches exist in isolation. Network analysis, for instance, might reveal that there are multiple distinct communities in a dataset: that, in technical terms, the network contains several clusters that are more strongly connected to each other than to the rest of the network. Having identified these clusters, we can then focus our attention specifically on them, and conduct a detailed qualitative or quantitative analysis for each cluster by itself. This may reveal, in our Twitter example, whether these groups of participants have different interests, attitudes, opinions, or activity patterns; depending on the context, such differences could distinguish one collective perspective from another, constructive from disruptive contributors, authentic participants from spammers and trolls, and so on. Network analysis is not just a tool in itself, therefore, but part of a broader toolkit that should contain a mix of qualitative and quantitative methods.

Basic concepts in network analysis

Before we get to those clusters, though, we need to define some of the more basic building blocks of networks. To begin with, networks consist of *nodes*: these are the entities that are connected with each other. Depending on the context of the network analysis, those nodes could be people, companies, concepts, media objects, and many other things; in our Twitter example, those nodes may be the Twitter accounts that contributed to a particular dataset, but they could also be the individual posts (tweets) themselves. In fact, the nodes in a single network might well represent several different types of entities, representing a *hybrid* network: our Twitter network, for instance, could represent the accounts themselves as one type of node, and the hashtags to which they contributed as another type of node.

These nodes form an actual network, then, if they are connected in some way, and that connection is called an *edge*. Any node might be connected to others through multiple edges: an email account that engaged with multiple other accounts would have edges connecting it to each of those accounts, for instance; a Twitter account that used several hashtags would have edges to each of those hashtags. Depending on the network, edges may have a specific direction: if account A emails account B, this creates an edge from A to B, but if B does not respond to the email, there is no reciprocal edge from B back to A (fig. 1). This example would represent a *directed* network, but not all networks are directed: if edges simply indicate that two nodes are part of the same group, then they are always reciprocal, and the resulting network would be an *undirected* network.

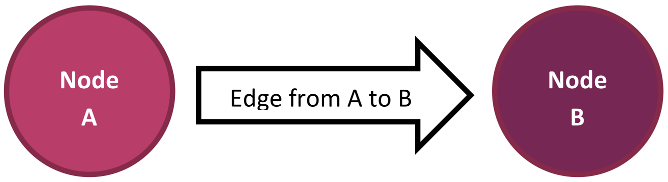


Fig. 1: A non-reciprocal connection from A to B, in a directed network.

Further, some networks may also include information about the strength of connections; two social media accounts that talk to each other all the time could be considered to be more strongly connected than two accounts that engaged with each other only once. This strength of connection is described as the *weight* of the edge between the two nodes (and in a directed network, the edge from A to B and the edge from B to A might have different weights). In very complex networks, we might even need to distinguish different types of edges between the same nodes, and calculate different weights for each edge type: for instance, A and B might be connected by email, tweet, direct message, face-to-face conversation, etc., and each of these connections could be treated as a separate edge type.

Finally, then, for each node in the network we can already calculate some basic attributes. To begin with, we can count how many other nodes it is connected to through its edges: this is the *degree* of the node. In a directed network, however, we must distinguish between incoming and outgoing edges: these are the *in-degree* and *out-degree* of the node. If an email account mailed three other accounts, and two of them mailed back, for example, then its out-degree is 3, and its in-degree is 2; its total degree is 3 + 2 = 5. Additionally, we can take into account not just the number of edges to other nodes, but also their weight: this produces a *weighted degree*, *weighted in-degree*, and *weighted out-degree* for each node. If the email account mailed each of the three other accounts three times, then its weighted out-degree is 3 + 3 + 3 = 9; if the two responders sent just one email each, then its weighted in-degree remains 1 + 1 = 2. The node's overall weighted degree would be 9 + 2 = 11 (fig. 2).

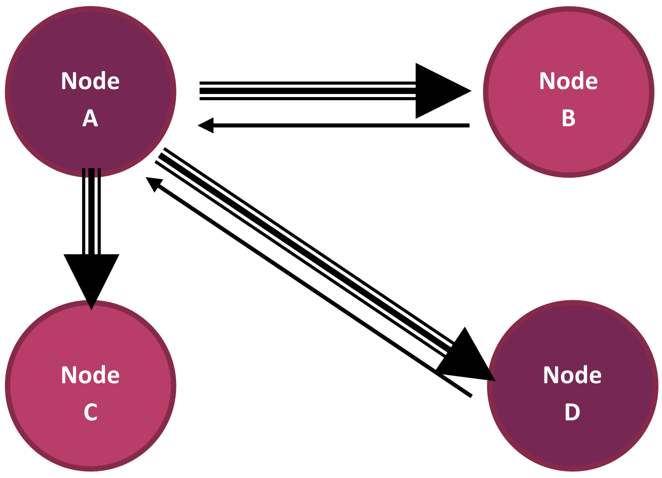


Fig. 2: Degree and weighted degree in a directed graph. A has an out-degree of 3 (it connects to B, C, and D), and an in-degree of 2 (B and D connect back to A), for a total degree of 5; but a weighted out-degree of 9 (it connects three times each to B, C, and D) and a weighted in-degree of 2 (B and D connect back only once each to A), for a total weighted degree of 11.

Installing Gephi

There are many more complex metrics that describe network structures, but these basic concepts – node, edge, direction, weight, and degree – are sufficient for us to get started with some basic network analysis. For this, we will use Gephi, a powerful open-source software tool that is now widely used in network analysis (Bastian et al., 2009). Gephi is freely available in Windows, Mac, and Linux versions from http://gephi.org/, which also offers some basic guides and support documents; on Windows and Linux, you will also need to install the latest version of Java (freely available from http://java.com/) to enable Gephi to run. Follow the instructions at https://gephi.org/users/install/ to install Gephi.

Gephi is a versatile tool for network analysis, but if you expect to be working with large network datasets (of hundreds of thousands of nodes, or more) at a later stage, we need to make sure Gephi has access to the right amount of memory available. As a beginner, you can skip the following steps – but come back to them if Gephi requests greater memory access:

Gephi's memory use is controlled from a text file called *gephi.conf*, which is part of the Gephi installation. First, find out how much RAM is available in your computer; on Windows, you can do so for example by right-clicking on the *My Computer* icon on your desktop and selecting *Properties*; look for the 'Installed RAM' value.

Now we will edit *gephi.conf* to allocate that memory to Gephi. On a Mac, right-click on the Gephi application icon, select *Show Package Contents*, and find the *resources/gephi/etc* folder. In that folder, use the TextEdit application to open *gephi.conf*. On Windows, navigate to where Gephi was installed – this is likely to be in a folder called *C:\Program Files\Gephi-0.9.3\etc* or similar. Use NotePad to open *gephi.conf*.

In that file, you should see a line that begins with:

```
default_options="--branding gephi -J-Dsun.java2d.metal=true
```

We will leave all other options alone, but you should insert a value in for the maximum amount of RAM available in your computer, minus a small buffer. Multiply the amount of RAM in your computer by 1,000, then subtract a small amount; for example, in a computer with 16GB of RAM, you could go with 16 × 1,000 – 500 = 15,500. We'll insert this as a new option as follows:

```
default_options="--branding gephi -J-Xmx15500m -J-Dsun…
```

This tells Gephi to use up to 15.5GB of the available RAM. Make sure you retain the 'm' at the end of the value, and save the edited *gephi.conf* file under the original filename. Now start Gephi; if the software will not start, you may have allocated more RAM than you have available in your computer. If so edit *gephi.conf* again and try a slightly smaller Xmx value.

Getting started with network visualisation and analysis in Gephi

When you first start Gephi, it will open its default *Welcome* window; simply close that window for now and proceed directly to Gephi itself. The Gephi user interface (fig. 3) is highly configurable, but you will always have the three tabs *Overview*, *Data Laboratory*, and *Preview* at the top, and start in *Overview*; you will also see the *Appearance* and *Layout* boxes in the left sidebar, and *Context* as well as the *Filters* and *Statistics* tabs on the right. In the centre is the *Graph* box, where your network visualisation will appear; this is empty for now as we have not created a network graph yet.

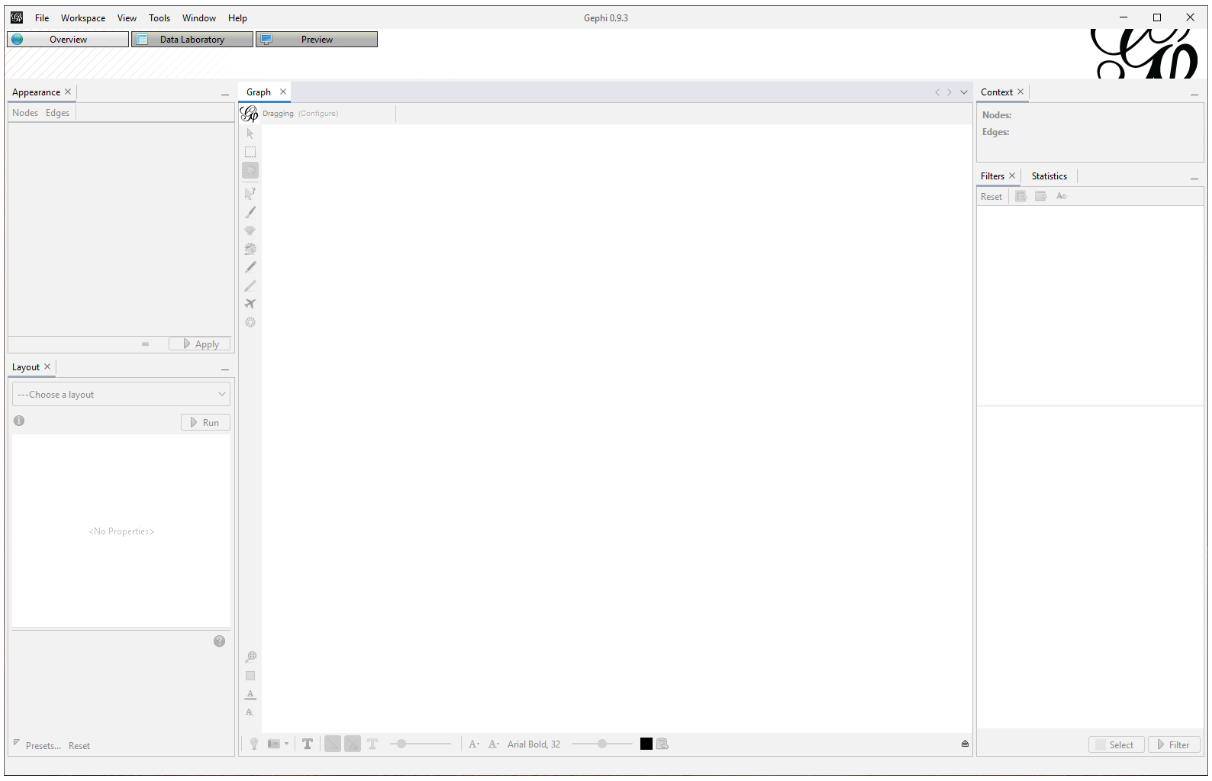
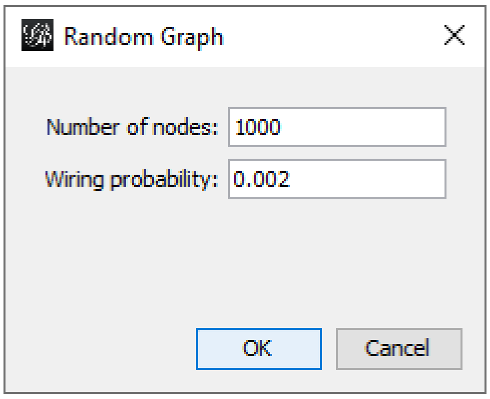


Fig. 3: The default Gephi user interface (on Windows)

After installation, it is a good idea to use the *Help > Check for Updates* menu first to see if there are any further updates to the current Gephi software; do this at regular intervals as you work with Gephi, too. Further, the *Tools > Plugins* menu provides access to a wide range of extensions and additions to the basic Gephi functionality; these add new network import, visualisation, analysis, and export options contributed by the Gephi community. In this guide we will work only with standard Gephi functionality, but explore these plugins at your own leisure and install the additional tools that may be useful for you. Also remember to check back regularly for further updates.

As we have no network data to work with yet, we will use Gephi to generate a random graph so we can explore its functionality. Use the *File > Generate > Random Graph…* menu to open the random graph generator, and generate a graph with 1000 nodes and a wiring probability of 0.002 (fig. 4).

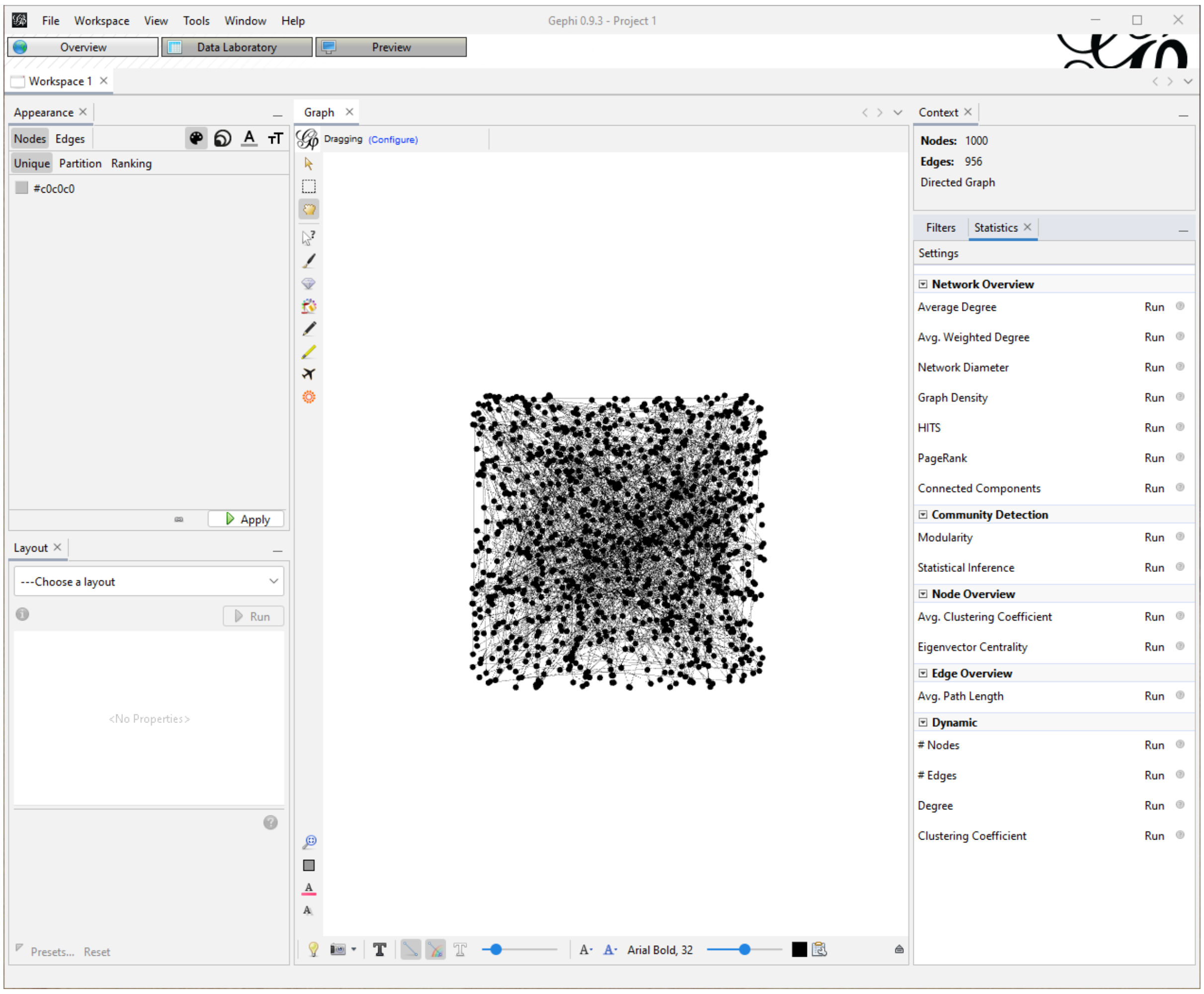

Random Graph

Number of nodes: 1000

Wiring probability: 0.002

Fig. 4: The dialogue box for creating random graphs, with the settings used for this exercise.

You will note that this has created a *Workspace 1* tab below the tabs at the top of the window, and an irregular square in the centre of the *Graph* box (Fig. 5) – this is the random network we have just generated. The network looks like this for now because the nodes have simply been positioned randomly across the available space, without taking into account the underlying structure of the network itself. If you move your mouse pointer across this graph, you will see that Gephi highlights the nodes and edges in the network that you are currently pointing to.



Nodes: 1000
Edges: 956
Directed Graph

Fig. 5: The default Gephi user interface, showing a random network graph.

Before we properly visualise this network, we need to calculate some basic network statistics. Select the *Statistics* tab in the right sidebar, and click on the *Run* buttons for *Average Degree* and *Avg. Weighted Degree* (fig. 6); this generates various statistics for the overall network, as well as for each individual node within the network. Gephi will show these statistics in new pop-up windows, which you can close once you have reviewed them (you can open them again with the *(?)* button next to *Run*). Note that for the purposes of this exercise, the values for degree and weighted degree are identical: Gephi's random graph generator only produces edge with a standard weight of 1.

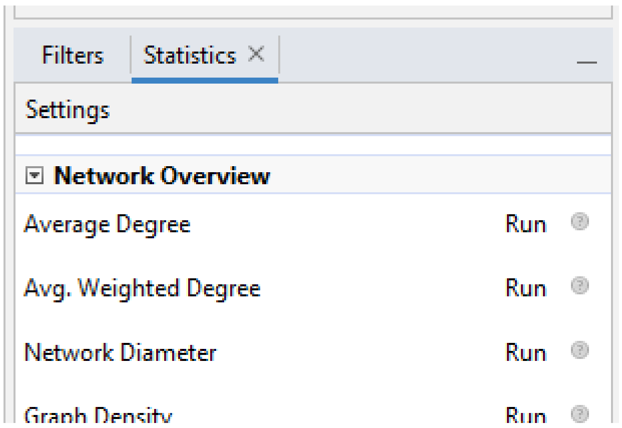
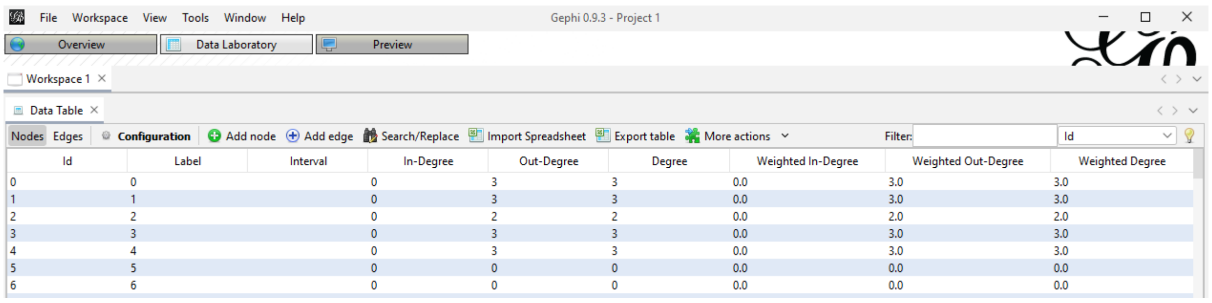


Fig. 6: The functions for calculating Average Degree *and* Avg. Weighted Degree.

By clicking on the *Data Laboratory* tab at the top of the window, you can see these statistics for each node: the *Data Laboratory* shows the data that the network is built from. The *Data Table* box in turn contains tabs for both *Nodes* and *Edges*: for nodes, it lists their (randomly generated) ID and label, as well as the various weighted and unweighted degree, in-degree, and out-degree values (fig. 7); for edges, it shows the ID of the source and the target of each edge, the randomly generated ID of the edge, the type of the edge (the edges created by the random graph generator are directed), and the weight of the edge (all 1.0 in this case). You can sort these tables by these values by clicking on the column headers, for example to identify the most or least connected nodes in the network. Finally, return to the *Overview* tab.



Id	Label	Interval	In-Degree	Out-Degree	Degree	Weighted In-Degree	Weighted Out-Degree	Weighted Degree
0	0		0	3	3	0.0	3.0	3.0
1	1		0	3	3	0.0	3.0	3.0
2	2		0	2	2	0.0	2.0	2.0
3	3		0	3	3	0.0	3.0	3.0
4	4		0	3	3	0.0	3.0	3.0
5	5		0	0	0	0.0	0.0	0.0
6	6		0	0	0	0.0	0.0	0.0

Fig. 7: Excerpt from the Data Laboratory *table of nodes in a random network.*

Visualising a network

Now we will begin the visualisation process. First, we will assign sizes and colours to the nodes in our network. Such visual aids are a crucial tool in network visualisation: we can use them to highlight specific properties of nodes (and edges) that are of particular importance to the analysis. What these properties are will vary from analysis to analysis, but for now let us use colour to highlight the nodes with the greatest in-degree (if our random network represented acts of communication, these would be the most important receivers of information), and size to highlight the nodes with the greatest out-degree (the most active senders).

In the *Appearance* box in the left sidebar, select *Nodes*, click the paint palette icon, select *Ranking*, and choose *In-Degree* as the metric to use. To the right of the *Color* scale, a small icon offers several default and recent colour options; if you move your mouse pointer over the *Color* scale, several triangles appear that enable you to shift the start, mid, and end points of the colour gradient. If you double-click on one of the triangles, you can adjust the colour at this point. In the example in fig. 8, the nodes with the lowest in-degree will be yellow, and those with the highest in-degree red. Click the *Apply* button at the bottom of the box to apply your colour scheme when you are ready; you should notice your network graph changing colour.

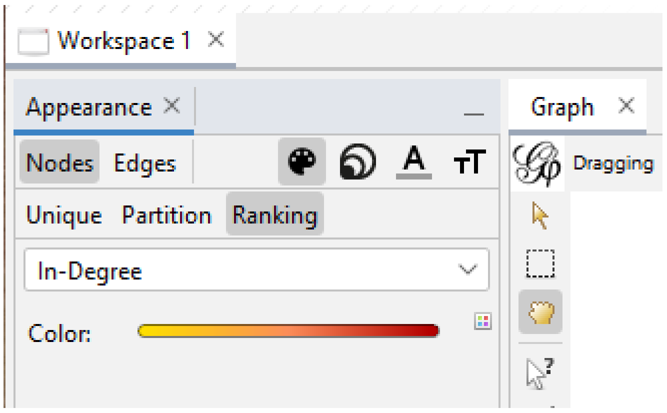
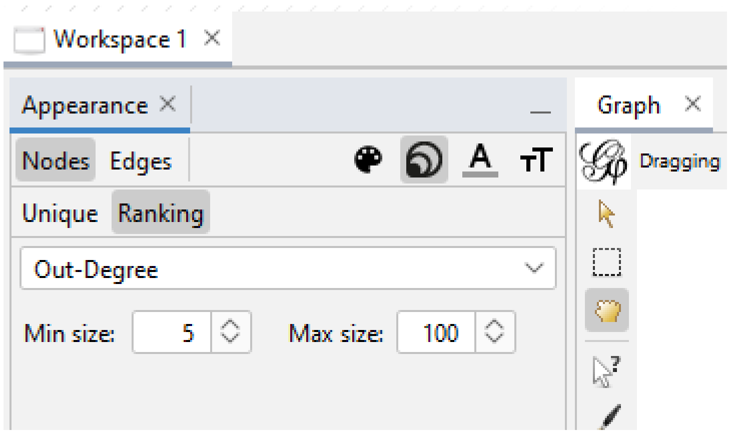


Fig. 8: Applying a colour gradient to the nodes.

Next, we will use the out-degree metric to determine the size of the nodes in our network. In the *Appearance* box, select *Nodes*, click the circles icon, select *Ranking*, and choose *Out-Degree* as the metric; then set a size range for the nodes in the network (there are no correct or incorrect values here; the example in fig. 9 uses a range from 5 to 100 to produce clear differences between the least and best connected nodes). Again, click *Apply* to activate these settings.



Out-Degree

Min size: 5

Max size: 100

Fig. 9: Applying a size gradient to the nodes.

The graph will now shore more structural differences; for instance, with the settings above, small red nodes would have a high in-degree and a low out-degree (if the graph represented email traffic, for example, these would be accounts that receive many and send few emails), while large yellow nodes have a low in-degree and a high out-degree (sending many, receiving few emails). You could apply these and other metrics to the graph in other combinations as well, of course (using size for in-degree and colour for out-degree, for example), so it is critically important to document these choices for every graph you create.

The network metrics you choose to highlight by using node colour and size depend on the aims of your network analysis; through its in-built statistics and the wide range of additional plugins available, Gephi offers a large number of metrics well beyond basic degree calculations. (See, for instance, the various centrality and eccentricity measures available through the *Network Diameter* statistics option in the right sidebar.) Many of these measures are described in detail in the relevant literature, and their use depends on what properties of your network you want to highlight: the most active or most targeted nodes; the nodes that are most central to connecting the entire network; or the nodes that are on the fringes and are most vulnerable to disconnection. You may also want to import additional attributes from your source dataset and use these metrics to colour or size the nodes in your network (e.g. size nodes by the number of followers a Twitter account has, or colour nodes by the country a user is based in). These choices must be made, documented, and justified on a case-by-case basis.

You may have noticed that the network edges have also been coloured; this is because by default, Gephi colours edges by the colour of the source node (in a directed network, the node from which the edge originates). We can switch this off by clicking the edge colour icon at the bottom of the *Graph* box, looking like a line with a rainbow (fig. 10).

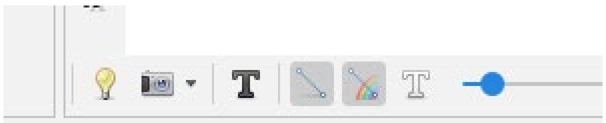
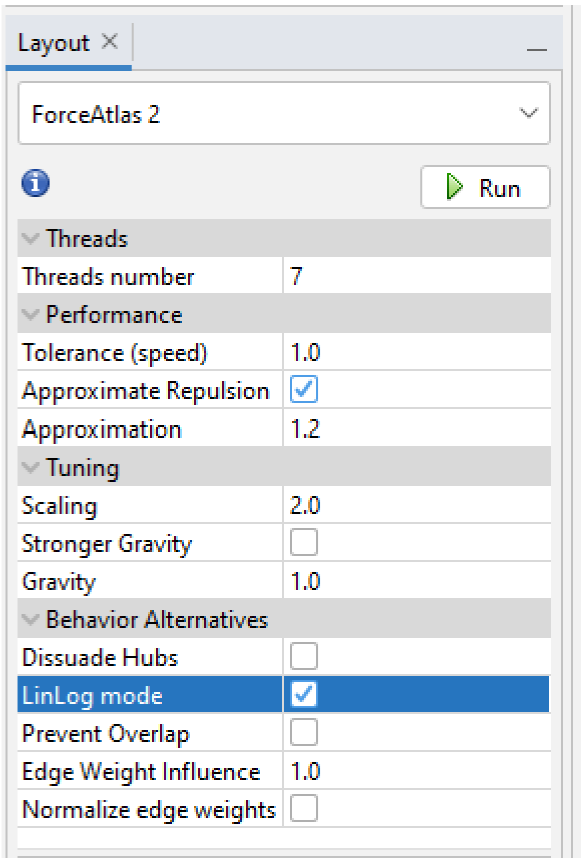


Fig. 10: The edge colour icon is the fifth in the icon row at the bottom of the Graph *box.*

Finally, we can now move on to the main network visualisation step: positioning the nodes in the network in relation to each other. There are a many different options for network visualisation, any Gephi itself offers a range of layout algorithms, but common to most of them is an underlying logic that positions the nodes based on how strongly they connect with others. This is often described as a simulation of a system of springs, or of gravity: nodes that are connected with each other through an edge pull each other closer, while nodes that are unconnected repulse one another; that attraction or repulsion may also be heightened by the weight of individual network edges, of course. By calculating the level attraction and repulsion for each pair of nodes in the network, the layout algorithm then attempts to find the place on the screen (or in some cases, in a three-dimensional space) where these forces are most balanced.

We can see this process in action as we visualise our network in Gephi. Here, we will use one of its most popular layout algorithms, Force Atlas 2 (Jacomy et al., 2014). In the *Layout* box in the left sidebar, select *Force Atlas 2* from the pull-down menu, choose a *Scaling* value of 2.0, and select the *LinLog mode* checkbox. Leave all other settings as shown in fig. 11 (but note that *Threads number* may be different for your computer – it identifies the number of processor cores you have available, and should be set to the maximum number minus 1). Click *Run* when ready.



Threads	
Threads number	7
Performance	
Tolerance (speed)	1.0
Approximate Repulsion	✓
Approximation	1.2
Tuning	
Scaling	2.0
Stronger Gravity	☐
Gravity	1.0
Behavior Alternatives	
Dissuade Hubs	☐
LinLog mode	✓
Prevent Overlap	☐
Edge Weight Influence	1.0
Normalize edge weights	☐

Fig. 11: Settings for the Force Atlas 2 layout algorithm.

Once you click *Run*, you will see the nodes in your graph move around each other as the layout algorithm attempts to find the optimal position for each of them; you may need to use the mousewheel to zoom out and view the full graph. For larger networks, this layout process may take some time, but eventually you will see the network settle into a fairly stable pattern with only minimal further movement (the algorithm will keep trying to improve the positioning of the nodes until you tell it to stop).

You can continue to use the mousewheel to zoom in and out to explore the network graph; you can also move your view around the graph by right-clicking and dragging anywhere in the *Graph* box. To centre your view on the full graph again, use the looking-glass icon at top of the icon bar in the bottom left-hand corner of the *Graph* box (fig. 12).

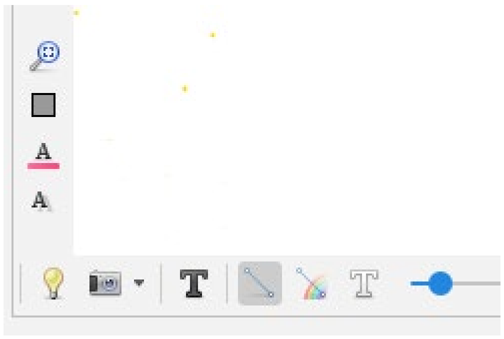
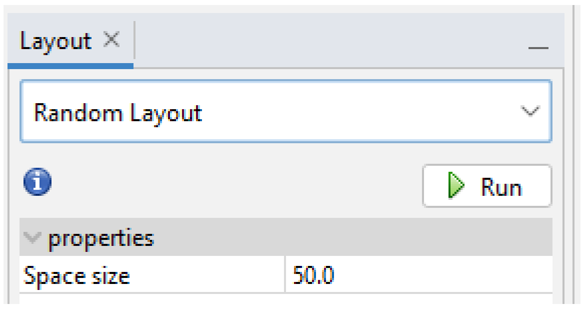


Fig. 12: The looking-glass icon returns the view to the entire graph. The lightbulb icon changes the background colour. The slider changes the edge thickness.

While *Force Atlas 2* is running, you can also try different settings for the *Scaling* value (e.g. 1.0, 3.0, 4.0 etc.) – this condenses or expands the graph, and can be useful for revealing further structural detail. However, if your *Scaling* value gets too large, you may see the outer nodes in the graph get stuck to the edges of the available display space – reduce the *Scaling* value, or click *Stop* on *Force Atlas 2*, change the visualisation option in *Layout* to *Random Layout* and click *Run* to reset the graph (fig. 13), and then start *Force Atlas 2* again.



Layout

Random Layout

Run

properties

Space size	50.0

Fig. 13: The Random Layout *option can be used to reset the graph visualisation.*

If you find the graph difficult to view with the colours you have chosen, left-click on the lightbulb icon in the bottom left-hand corner of the *Graph* box to switch between white and black background, or right-click on the lightbulb to choose a different background colour. You can also use the slider next to the *T* icon to adjust the thickness of the network edges (fig. 12).

Identifying network clusters

When we started, we used a low *Wiring Probability* value to create a relatively sparse network with few edges between nodes, but because it was randomly generated, our network is still going to be fairly uniform in shape (such networks are often referred to as 'furballs', because of how they appear on screen); networks that represent real-world data may show more distinct internal structures. You may wish to repeat the previous steps a few times to see if they result in a network with distinct clusters of highly connected nodes within an otherwise sparse network graph. This will be useful for our next steps.

By the end of the preceding steps your network may look something like the example shown in fig. 14: from a dense centre, several lobes of less connected nodes extend towards its edges, and the network core is surrounded by a number of single nodes that are disconnected from the centre.

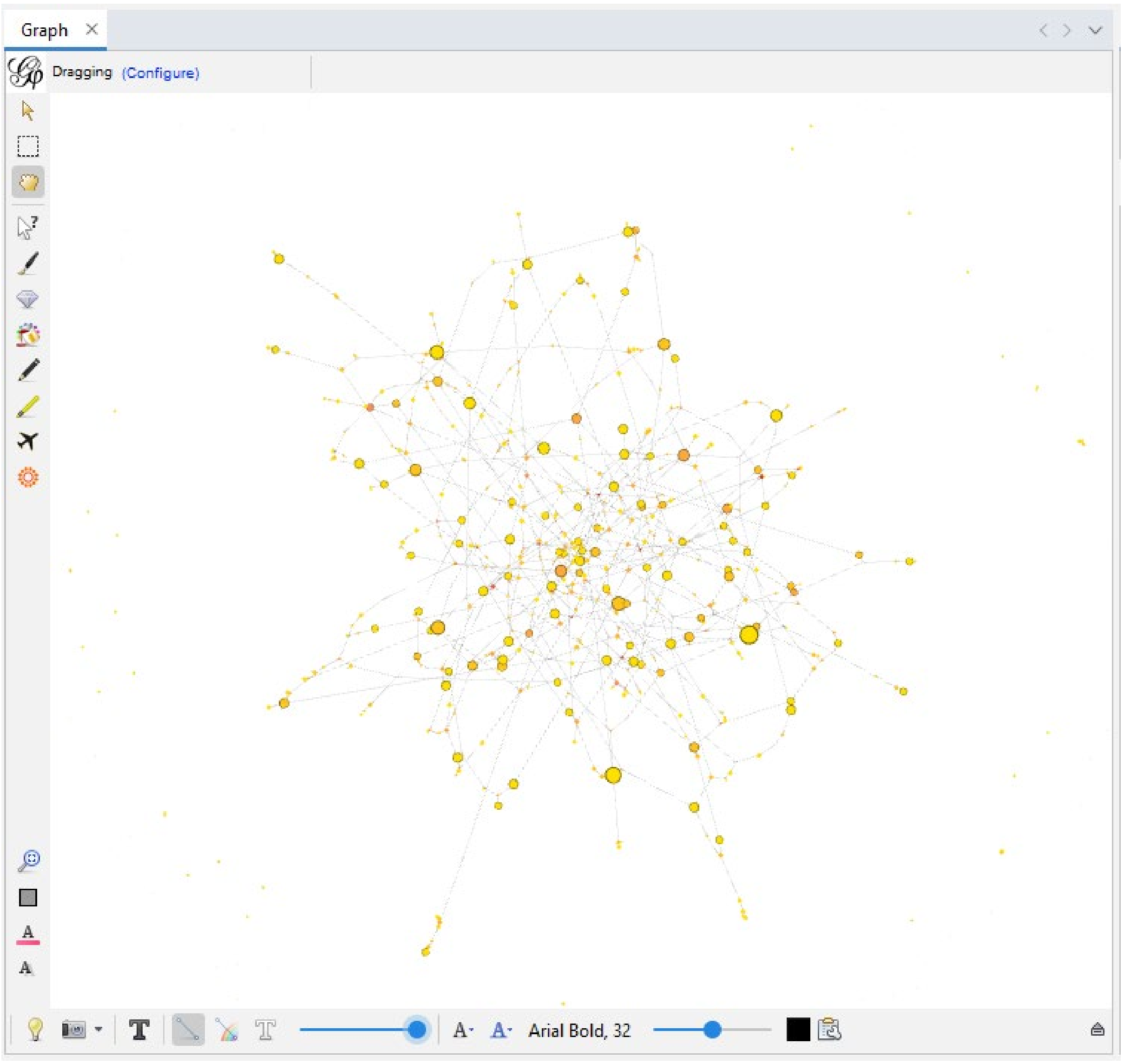
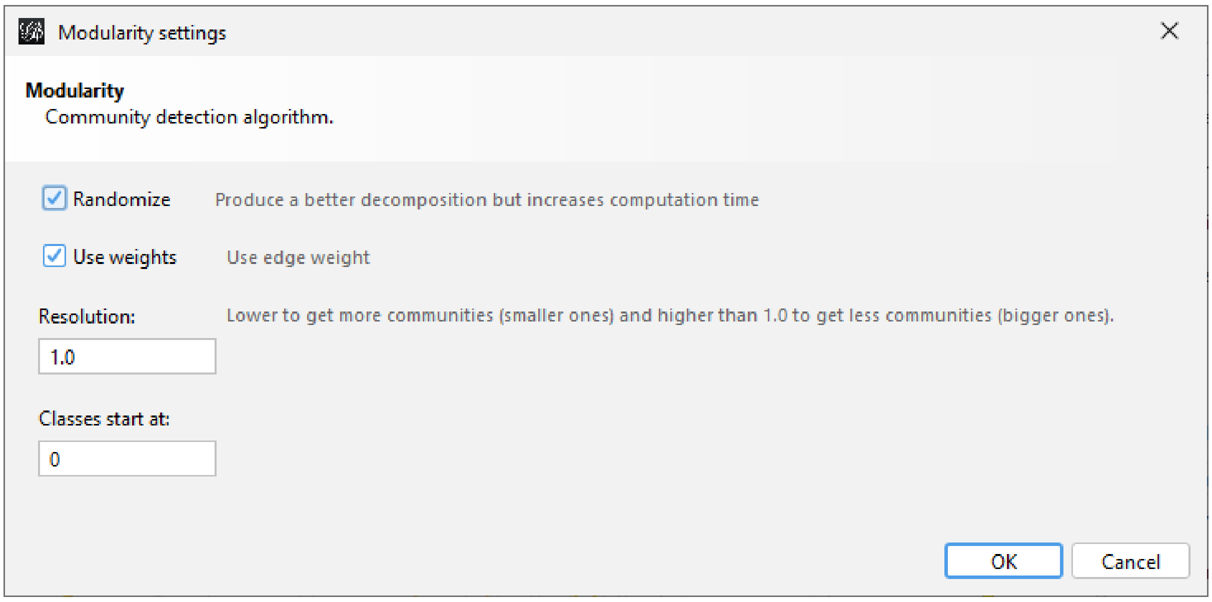


Fig. 14: A random network generated through the preceding steps.

We can add to this visual description of the network structure by applying a more systematic cluster detection algorithm to the network. To do so, Gephi provides a function called *Modularity*, in the *Statistics* box in the right sidebar (the algorithm used here draws on Blondel et al., 2008). Clicking on *Run* next to *Modularity* opens the dialogue box in fig. 15:


Modularity settings

Modularity
Community detection algorithm.

☑ Randomize — Produce a better decomposition but increases computation time

☑ Use weights — Use edge weight

Resolution: 1.0 — Lower to get more communities (smaller ones) and higher than 1.0 to get less communities (bigger ones).

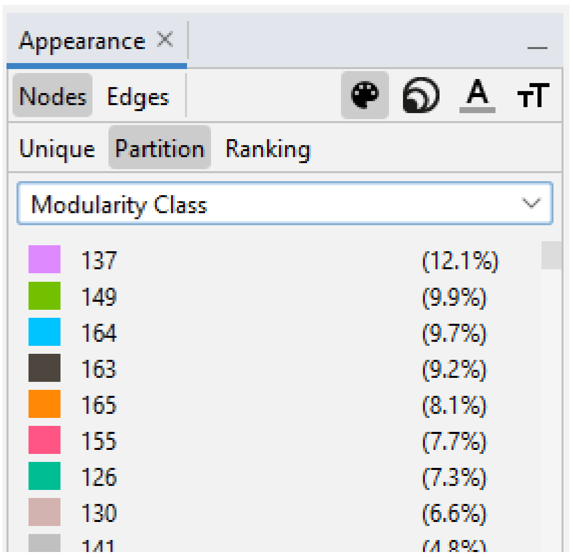
Classes start at: 0

Fig. 15: Modularity detection settings.

Leave *Randomize* and *Use Weights* ticked (but note that our sample graph does not have weighted edges, so this option will make no difference here). The key value we need to focus on is *Resolution*, which controls whether the algorithm will produce many small clusters or a few large clusters: in essence, smaller values for *Resolution* mean that the algorithm is more sensitive to small differences in the network structure.

Because our randomly generated graph is not particularly realistic and has limited internal structure, we can use a relatively high value (try 3.0 to begin with); real-world data may produce better results for values of 0.5 or below. It is important to note that there is no one ‘correct’ *Resolution* value: it simply controls the modularity algorithm’s sensitivity. The values you choose in your visualisations of real (non-random) data depend on the nature of your data, and it may well be useful to present the results of modularity detection at multiple levels of resolution, in order to identify broader clustering patterns (using high *Resolution* values) and then identify more specific sub-clusters within these larger clusters (using low *Resolution* values).

For now, set *Resolution* to 3.0, click *OK*, and wait for the results window to show; then close that window. We will add the results of this modularity detection process to our graph by changing the node colours to reflect the network cluster they belong to. In the *Appearance* box in the left sidebar, select *Nodes* and the colour palette icon, and click on *Partition*. From the pull-down menu that appears, choose *Modularity Class* as the attribute to be used in partitioning the network: *Modularity Class* is a numerical ID that the modularity algorithm assigns to all nodes belonging to the same cluster. In the *Appearance* box, you should now see a list of these classes, and an indication of what percentage of nodes in the total network belong to each class (fig. 16; your IDs and percentages will be unique to your graph, of course).



Modularity Class

🟪	137	(12.1%)
🟩	149	(9.9%)
🟦	164	(9.7%)
⬛	163	(9.2%)
🟧	165	(8.1%)
🟥	155	(7.7%)
🟩	126	(7.3%)
🟫	130	(6.6%)
⬜	141	(4.8%)

Fig. 16: A listing of modularity classes and their sizes in the Appearance *box.*

To visualise these in the network, we now need to apply a colour scheme to these classes. At the bottom of the *Appearance* box, click on the *Palette…* option (fig. 17). This shows any recently used colour options (this will be empty if you have not used this function before), provides some standard options from the *Palettes >* function, and enables you to create new colour palettes via the *Generate…* function. Click on *Generate…* now.

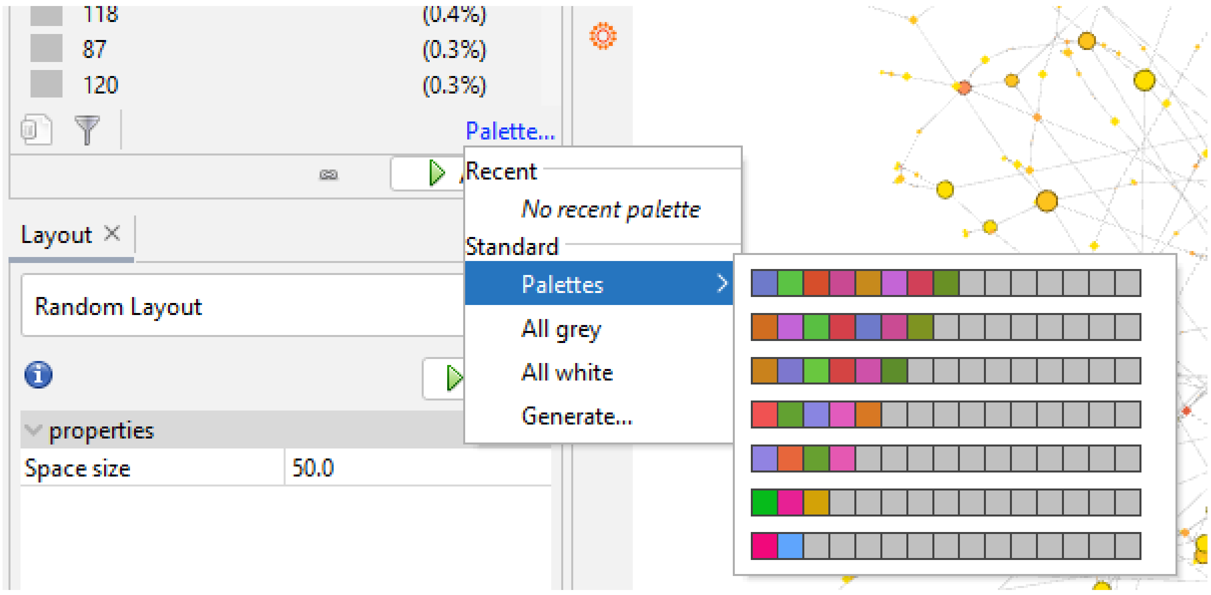


Fig. 17: Generating new colour palettes for cluster visualisation.

The dialogue box that appears now shows how many clusters there are in your network (*Values count*; 210 in fig. 18), but as many of these will be very small clusters consisting of one or two nodes only, not all of them need to be identified by colour. Your list of clusters in the *Appearance* box is likely to show that only 10 to 15 clusters account for more than 1% of the network; we can focus on these in our visualisation. Therefore, keep *Limit number of colours* ticked and enter 15 as the number of distinct colours to generate; select a colour scheme from the drop-down menu (but note that there is an error in the scheme names: *Fancy (light background)* is actually best suited to a dark background, and vice versa), and click *Generate*. You can repeat this step again if you don't like the resulting palette, for example because the colours it contains are too similar to each other.

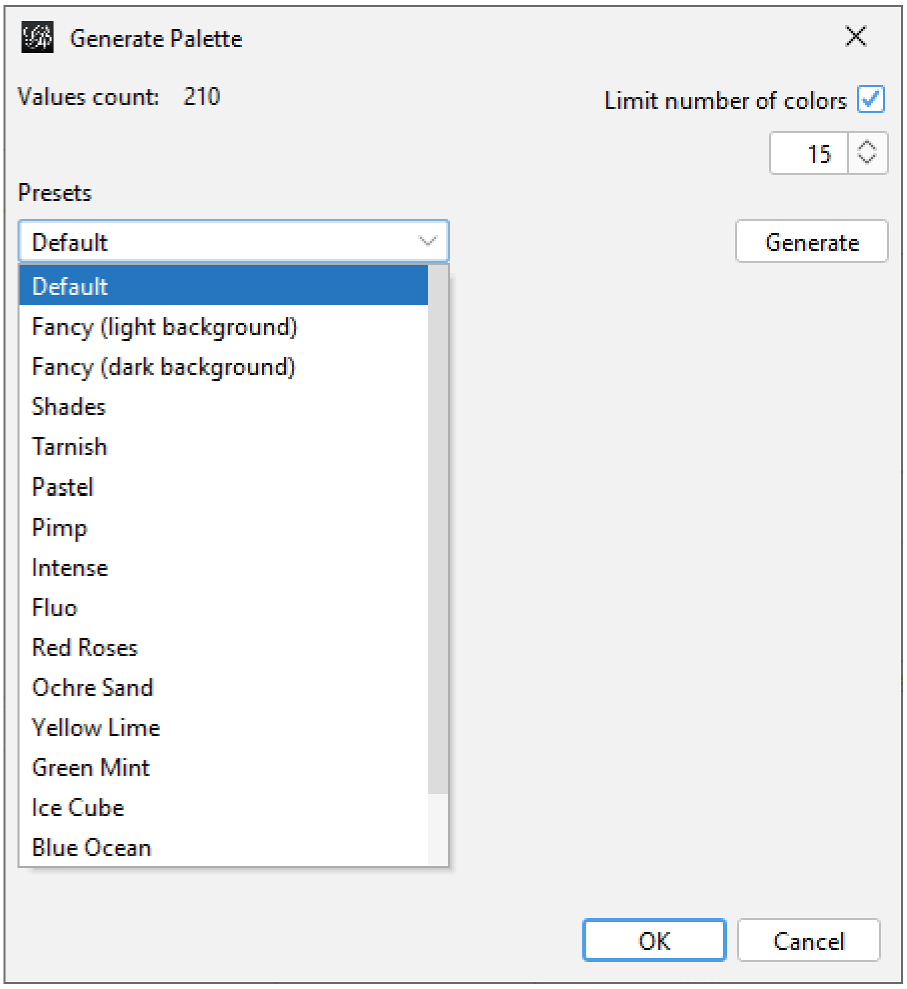


Fig. 18: The dialogue box for generating a new colour palette.

The new colour scheme will already have been applied to the cluster list in the *Appearance* box; to apply it to the graph itself, click the *Apply* button. Note that any clusters beyond the 15 generated colours will retain their grey default colour.

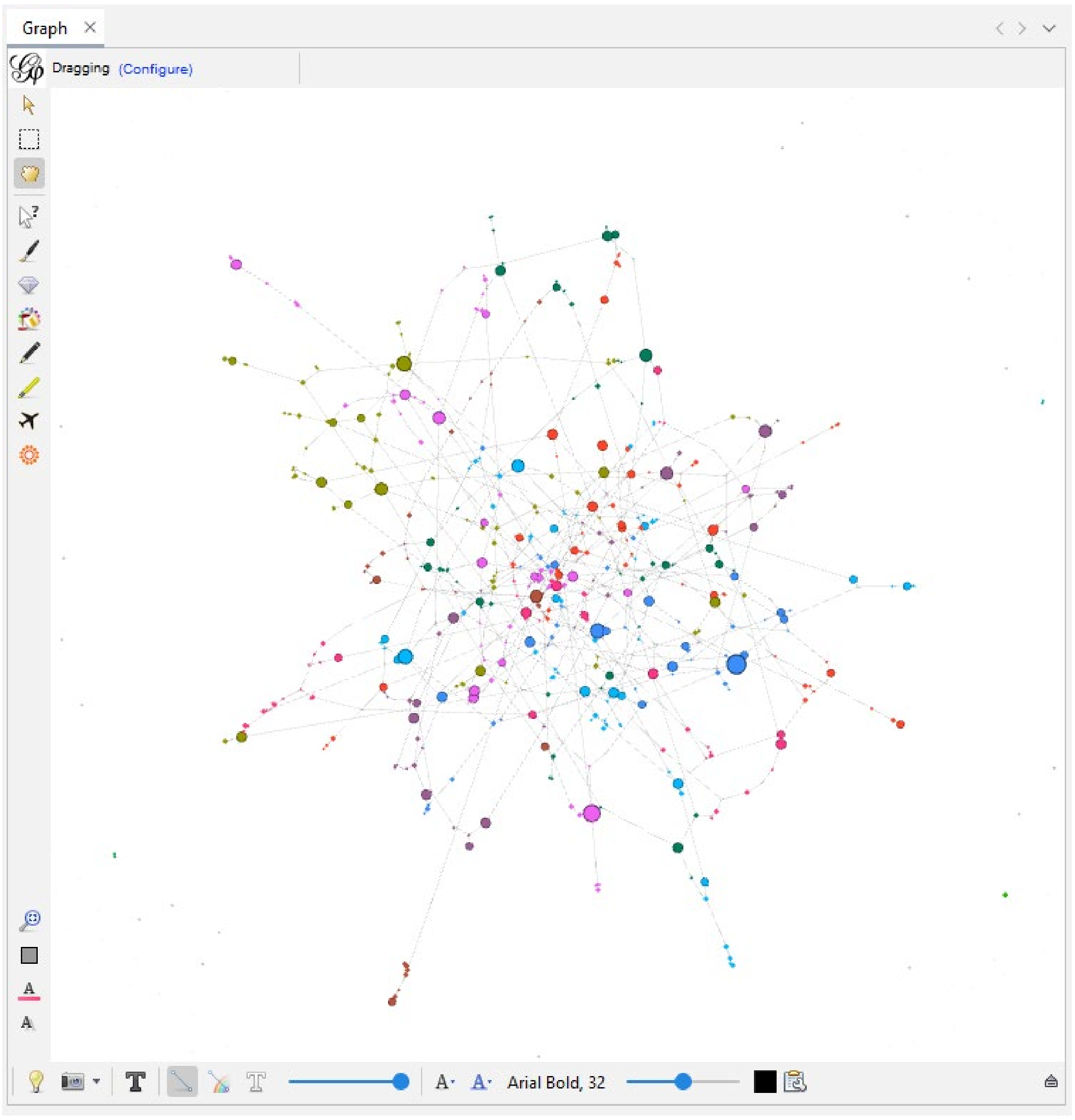


Fig. 19: A fully visualised random graph, with clusters indicated by node colouring.

The resulting graph may confirm some of the observations from your initial visual inspection, with more densely connected regions showing the same colours, and distinct clusters coloured differently (fig. 19). But it is important to note here that there is no one absolute truth that you can extract through this cluster detection process: running the modularity algorithm with different sensitivity settings will produce very different results, from adding all nodes to the one giant cluster to assigning every node its own cluster of one. Indeed, for larger networks it may be useful to present these higher- and lower-sensitivity versions side by side, to show the overall network structure *and* the further subdivisions that may exist within the network's larger clusters (fig. 20).

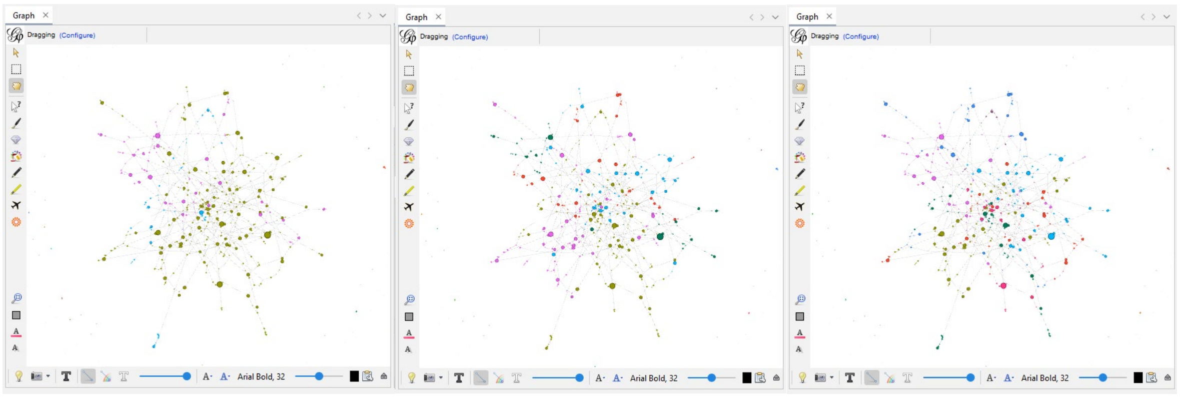


Fig. 20: A random graph visualised at modularity resolutions 7.0 (left), 5.0 (centre), and 3.0 (right), showing increasingly distinct substructures in the overall network structure.

The algorithmic analysis and your own visual inspection of the network should work hand in hand, therefore: if the modularity detection suggests cluster distinctions that do not seem to be supported by the visual network structure, you may need to reduce the algorithm's sensitivity; if it joins together parts of the network that appear distinct in the visualisation, you may need to increase it. The central point here is that network visualisation is always predominantly a tool for the *qualitative interpretation* of network structures, rather than for producing objective and absolute results in its own right.

Conclusion

This guide has stepped you through some of the core principles and processes in network visualisation and analysis, using the open-source tool Gephi for the practical exercises. Gephi has a substantial number of other, powerful analysis and visualisation features: in addition to *Force Atlas 2*, it offers a range of further visualisation options, both built into the core software and available as additional plugins; it provides many other functions for producing network statistics that can be used in the network visualisation; it enables you to filter the overall network to highlight only those components that are particularly notable or important for your analysis; it has advanced functionality for importing network data and exporting visualisations; and it even offers functions for visualising the change of network structures over time (Bruns, 2012).

It is impossible to cover all of these functions in a brief guide, and you may want to explore them further by engaging with the resources compiled by the Gephi user community at *gephi.org*, or joining the Gephi users group on Facebook. There are also various textbooks on network visualisation in general, or social network analysis in particular, that will offer further advice on specific aspects of this work. Above all, though, it is important to experiment, as network visualisations must always be sensitive to the specific networks they visualise – and as you do so, make sure you keep detailed notes (including screenshots) of the specific visualisation settings you use, both so that you can retrace your steps when revisiting the analysis at a later stage and so that you can document and defend these settings when you present or publish the research that draws on these visual analyses.

Further Reading

Robins, G. (2015). *Doing social network research: Network-based research design for social scientists*. Sage.
Ackland, R. (2013). *Web social science: Concepts, data and tools for social scientists in the digital age*. Sage.
Scott, J., & Carrington, P. J. (2014). The SAGE handbook of social network analysis. Sage.
https://www.doi.org/10.4135/9781446294413
Rogers, R. (2019). *Doing digital methods*. Sage.

Web Resources

- Gephi Website: https://gephi.org/
- Gephi learners' resources: https://gephi.org/users/
- Gephi Facebook group: https://www.facebook.com/groups/gephi/

References

Bastian, M., Heymann, S., & Jacomy, M. (2009). Gephi: An Open Source Software for Exploring and Manipulating Networks. *Proceedings of the Third International ICWSM Conference*, 361–362. http://www.aaai.org/ocs/index.php/ICWSM/09/paper/viewFile/154/1009/

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, *2008*(10), P10008. https://doi.org/10.1088/1742-5468/2008/10/P10008

Bruns, A. (2012). How Long Is a Tweet? Mapping Dynamic Conversation Networks on Twitter Using Gawk and Gephi. *Information, Communication & Society*, *15*(9), 1323–1351. https://doi.org/10.1080/1369118X.2011.635214

Castells, M. (2007). Communication, Power and Counter-Power in the Network Society. *International Journal of Communication*, *1*, 238–266. http://ijoc.org/index.php/ijoc/article/view/46

Jacomy, M., Venturini, T., Heymann, S., & Bastian, M. (2014). ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. *PLOS ONE*, *9*(6), e98679. https://doi.org/10.1371/journal.pone.0098679

Katz, E., & Lazarsfeld, P. F. (1955). *Personal Influence: The Part Played by People in the Flow of Mass Communications*. Routledge. https://doi.org/10.4324/9781315126234