Interactive visualization of community structure in complex networks

Anton Eriksson

June 6, 2018
INTERACTIVE VISUALIZATION OF COMMUNITY STRUCTURE IN COMPLEX NETWORKS

June 6, 2018

Copyright © 2018 Anton Eriksson (anero164@student.umu.se)

Submitted as a master’s thesis in Engineering Physics to the Department of Physics, Umeå University.

Supervised by Martin Rosvall, Integrated Science Lab, Department of Physics, Umeå University. Examiner was Ludvig Lizana, Integrated Science Lab, Department of Physics, Umeå University.
Abstract

Several applied sciences model system dynamics with networks. Since networks often contain thousands or millions of nodes and links, researchers have developed methods that reveal and highlight their essential structures. One such method developed by researchers in IceLab uses information theory to compress descriptions of network flows with memory based on paths rather than links and identify hierarchically nested modules with long flow persistence times. However, current visualization tools for navigating and exploring nested modules build on obsolete software that requires plugins and cannot handle such memory networks.

Drawing from ideas in cartography, this thesis presents a powerful visualization method that enables researchers to analyze and explore modular decompositions of any network. The resulting application uses an efficient graph layout algorithm adapted with a simulation based on information flow. Like in a topographic map, zooming into the map successively reveals more detailed community structures and network features in a continuous fashion.

Keywords

network, visualization, clustering, community detection
Contents

1 Introduction 1
  1.1 Aim and limitations 2
  1.2 Roadmap 2

2 Theoretical background 3
  2.1 Community detection in networks 3
  2.2 Graph layout algorithms 5
    2.2.1 Highlighting information flow 5
    2.2.2 Precalculating layouts 6

3 Implementation 7
  3.1 Level of detail 9
  3.2 Revealing module structure 9
  3.3 User interaction 10

4 Results 13

5 Conclusions 15
  5.1 Future improvements 15

6 Acknowledgements 17

7 Bibliography 19
1

Introduction

Several applied sciences, for example immunology, paleontology and biology, use networks to model large connected systems. The study of networks is called network science. It uses ideas from graph theory, statistics, computing science and physics to describe real-world phenomena based on interaction data.¹

A network is a set of nodes and links (fig. 1.1). Each node represents some entity in the data and the links connect these entities. For example, in a social network, the nodes would be people and the links would be the relationships formed in different ways.

While data is abundant in our connected society, the challenge is making sense of networks consisting of thousands or millions of nodes and links. To better understand large networks, it is helpful to highlight the most important structures. A way of revealing structure is to employ a clustering algorithm that partitions the network into modules of smaller networks. Clustering simplifies the network and makes the underlying structure easier to see. Infomap is one such clustering algorithm, and it uses a random walk on the network as a model of the flow of information. The optimal way to describe a random walk on a clustered network is known as the map equation², and Infomap tries to minimize the map equation by partitioning the network into modules.

Powerful clustering algorithms such as Infomap do not take us all the way. To analyze, reason, and draw conclusions there is a need for visualization methods that reveal just the right amount of information – important features should be highlighted, and noise should be minimized.

An application called the Hierarchical Network Navigator already exists to visualize clustered networks.³ It uses two separate windows to navigate the network; one for selecting the module you want to look at, and another for displaying the contents of that module. (fig. 1.2). The Hierarchical Network Navigator is a mature application, but it has some limitations. It is developed using the web-technology Flash, which has decreased in popularity over the past several years and requires a web-browser plugin.
It also disconnects the hierarchical structure from the module structure. That is, when you look at the module structure, you cannot see how that module interacts with its neighbors.

To improve on this, we present a method inspired by Google Maps to integrate the hierarchical view with the module view into a single window. The idea is to think of the highest-level modules as the continents on a map. The analogue of countries is revealed when you zoom in. Further zooming reveals cities and streets, but the continents do not distract our view at this point. This way, we can see more than one level and module at a time which gives context and a clear overview.

1.1 Aim and limitations

The aim is to develop an application for the web written in JavaScript using modern development tools and frameworks, make it intuitively usable by researchers, and use a method of interaction similar to Google Maps. It should feel like "a product," meaning that it should present itself as a part of the applications developed by the team behind Infomap.

The application targets only the newest versions of the most widely used web-browsers, which means that it might work on older web-browsers but we do not test this. Furthermore, the application requires a reasonably fast computer with a good display. The upside of these constraints is that development time is focused on features rather than compatibility.

1.2 Roadmap

In the following parts we present an overview of networks, network clustering, and the map equation. Next, we adapt a graph layout algorithm in a way to highlight information flow. Then, we discuss the implementation of the new visualization method and a rank-based level-of-detail to reduce clutter and noise. We compare the resulting application to the Hierarchical Network Navigator using the same network data.

We conclude with ideas for future improvements.
A network is a graph \( G = (V, E) \) constructed from a set of nodes \( V \) and links \( E \), where the nodes and links\(^1\) can have certain properties.

Two nodes \( n_i, n_j \in V \) are connected if there is a link \( l \) between the nodes, that is, \( (n_i, n_j) \in E \). Links can be directed or undirected, meaning that they can be traversed only in one direction or in both directions. Links can have a weight that determines how strong or important the connection between two nodes is, or they can be unweighted which means that they have uniform weight.

The number of links connected to a node is called its degree \( k \). The in-degree \( k_{in} \) is the number of incident links to a node in a directed network. The out-degree \( k_{out} \) is the number of outgoing links (fig. 2.1).

The aim is to visualize community structure in networks. To make sense of this we need to know how to find community structure and how to visualize networks using graph layout algorithms.

### 2.1 Community detection in networks

A clustered network is a network partitioned into modules in different levels. A two-level clustering as in fig. 2.2 would mean that the network is partitioned into modules that contain nodes. If there are more than two levels\(^2\), modules could contain either nodes or other modules.

![Figure 2.1: The number of links connected to a node is called its degree \( k \). A has in-degree \( k_{in} = 1 \) and out-degree \( k_{out} = 3 \), while B has \( k_{in} = 1 \) and \( k_{out} = 2 \).](image)

\(^2\) Called multi-level.

To choose how many modules the network is partitioned into, and which nodes should go into each module, we need to know a bit about random walkers. A random walk on a network is where

\(^1\) Other common names are vertices and edges.

![Figure 2.2: The description length of a random walk produces community structure. From left to right: (left) the original network with the beginning of a random walk as a dotted line, (middle) the network with colored communities, and (right) the communities represented as modules. The figure is own work inspired by work by M. Rosvall.](image)
you choose a node as a starting point and let the walker jump to nearby connected nodes with a probability proportional to the link weight between the nodes.

The random walker tends to visit some parts of the network more than others. For example, a random walker spends more time in highly connected areas than jumping back and forth between two nodes that only share a weak link. If the random walker spends more of its time in parts of the network, and only rarely jumps between these parts, we can describe its path relative to the jumps. For example, looking at the middle figure in fig. 2.2, we could describe a path as

\begin{align*}
\text{(start in green)} \\
1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \\
\text{(switch from green to blue)} \\
1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \\
\text{(switch from blue to yellow)} \\
1 \rightarrow 2 \ldots
\end{align*}

and so on. The path is more efficiently described using Huffman coding than with words and colored numbers. With Huffman coding, the node with the highest visit frequency by the random walker gets the shortest code, which minimizes the average code length. If we have a network partition $M$ into $m$ modules, the map equation

\[ L(M) = q \cdot H(Q) + \sum_{i=1}^{m} p^i \cdot H(P^i) \tag{2.1} \]

specifies the shortest code that can describe the path of a random walk on that partition. That is, the map equation measures how effectively we can encode the path using as few bits of information as possible.

The first term in eq. (2.1) measures the number of bits used for switching between modules where $q \cdot H(Q)$ is the rate at which the index code book is used and $H(Q)$ is the frequency weighted average code length in the index code book. The second term measures the number of bits used for jumping between nodes within a module and the code for exiting the module, where $\sum_{i=1}^{m} p^i \cdot H(P^i)$ is the module code book use rate and $H(P^i)$ is the frequency weighted average code length in the module code book.

Infomap does community detection by minimizing the map equation – it tries to find a partition $M$ of the network into $m$ modules such that the description length of a random walk on that partition of the network is the shortest possible while avoiding overfitting.

---

5 We call this the highest ranking node.


2.2 Graph layout algorithms

There are many different graph layout algorithms available. Some have a high run time but produce good results with few overlapping links, others are fast but have poor local minima.

The application in this thesis is built around a force-directed graph layout implementation supplied with the visualization library D3\textsuperscript{7}. Force-directed graph layout algorithms are of the fast-but-poor kind, they model networks as systems of springs between the connected nodes\textsuperscript{8}, and a pairwise repulsive charge between all nodes (fig. 2.3). This makes the connected nodes want to stay at a fixed distance to each other, while all nodes try to spread out as much as possible. The system evolves over time until the forces settle into equilibrium at which a local energy minimum is reached.\textsuperscript{9}

![Diagram](image.png)

2.2.1 Highlighting information flow

To highlight the flow in the network, we could make the forces between the nodes depend on the flow. For example, to let the flow $f_{ij}$ between nodes $n_i$ and $n_j$ influence the rest length $d$ of the spring such that higher flow results in shorter springs,

$$d_{ij}(f_{ij}) = -\frac{d_{\text{max}} - d_{\text{min}}}{f_{\text{max}} - f_{\text{min}}} f_{ij} + \frac{d_{\text{max}} f_{\text{max}} - d_{\text{min}} f_{\text{min}}}{f_{\text{max}} - f_{\text{min}}},$$

where $d_{\text{min}}$ and $d_{\text{max}}$ are the desired minimum and maximum rest lengths, $f_{\text{min}}$ and $f_{\text{max}}$ are the maximum and minimum flow between all nodes.

Furthermore, links with a higher flow can yield a stiffer spring. We use the spring strength

$$k_{ij} = \left(1 - \frac{0.5}{f_{\text{max}} - f_{\text{min}}} f_{ij} + \frac{0.5 f_{\text{max}} - f_{\text{min}}}{f_{\text{max}} - f_{\text{min}}}ight) b_{ij},$$

$$b_{ij} = \frac{1}{\min\{k_{\text{in}}, k_{\text{out}}\}},$$

where the first term weakens springs between nodes in proportion to the decrease in flow, and $b$ is a bias that weakens links to heavily connected nodes. The bias term is the default behavior in the library implementation.

The result of the above modifications are stiffer, shorter springs between nodes with high flow, and softer, longer springs between nodes where there is low flow. This highlights strong connections and makes weak connections less influential of the overall layout.

\textsuperscript{7} www.d3js.org

\textsuperscript{8} Similar to how you would simulate an elastic solid as a system of particles.


Figure 2.3: A graph made up of four nodes in a force-directed layout. Nodes are connected by springs (zigzag lines) which has an attractive force, and charges (dashed lines) which has repulsive force. The layout is achieved when the forces settle into equilibrium.
2.2.2 Precalculating layouts

To avoid showing the initial chock the nodes feel when they are subjected to forces from springs and potentials, we must hold off updating the layout until the simulation has cooled. But waiting until equilibrium also has drawbacks as the goal is a highly interactive application – we want to wait as little as possible. This means a compromise between waiting for the decreasing of the forces and showing the layout as soon as possible.

The number of iterations $N$ allowed to reach an equilibrium can be adjusted using a damping factor $\alpha$, with $0 < \alpha \leq 1$. The forces applied to each node are multiplied by the damping factor, which decays as $1 - \alpha_{\text{min}}^{1/N}$ where $\alpha \leq \alpha_{\text{min}}$ is the stopping condition. The damping factor for iteration $i$ is

$$\alpha_i = \alpha_{i-1} - \alpha_{i-1} \left(1 - \alpha_{\text{min}}^{1/N}\right)$$

$$= \alpha_{i-1} \alpha_{\text{min}}^{1/N},$$

which with $\alpha_0 = 1$ is the same as

$$\alpha_i = \alpha_{\text{min}}^{i/N}, \quad \alpha_N = \alpha_{\text{min}}.$$  \hspace{1cm} (2.6)

Using $N = 100$ iterations and $\alpha_{\text{min}} = 0.001$ results in the decay in fig. 2.4. In an interactive application it is preferred to show the layout as soon as possible, that is, when the damping factor is small enough, for example when $\alpha_i = 0.2$. Solving eq. (2.6) with $N = 100$ and $\alpha_{\text{min}} = 0.001$ for real $i$

$$0.001^{1/100} = \alpha_j = 0.2$$

$$\Rightarrow \quad i = 100 \frac{\ln 0.2}{\ln 0.001} \approx 23 \text{ iterations.}$$

This means that waiting 23 iterations before updating the layout, the forces on the nodes will have decreased by 80 percent.
3 Implementation

Having done community detection on a network with Infomap, it is time to visualize it. The output of Infomap is a text file which describes the module assignments of the nodes and the resulting code length. There are several output formats to choose from. The most suitable format for visualization is ftree since it contains all necessary information.

The ftree output format has the structure

```
*Nodes 1000
# path flow name id
1:1:1 0.0564732 "Name 1" 29
1:1:2 0.0066206 "Name 2" 286
1:1:3 0.0025120 "Name 3" 146
(more lines...)
*Links directed
# path exitFlow edges children
*Links root 0 68 208
1 2 0.000107451
2 1 0.0000830222
(more lines...)
# path exitFlow edges children
*Links 1 0.002 40 23
1 2 0.000042
2 1 0.000040
(more lines...)
```

where we first have a section describing the module assignment of each node, followed by several sections containing links, one section per module. This needs to be translated into a data structure that is suitable for visualization.

There are several ways to represent things like networks, modules, nodes and links as data structures. Networks can be considered a special case of a module. Both modules and nodes have a parent module and a module assignment. All of them contain flow, but only some have outgoing flow.

Instead of trying to create a taxonomy of classes, there are only the classes Network, Node and Link, without using inheritance. Instead, they share properties through object composition. Shared properties are extracted into separate behavior components that are composed into the base object at construction (fig. 3.1).
After parsing the Infomap output to an object representation, we need to display it. A good choice is to use Scalable Vector Graphics, SVG, a well-supported vector format. SVG can be infinitely scaled without losing resolution and can display shapes such as circles, squares, quadratic curves, etc.

Using SVG, we represent nodes and modules as circles whose area depends on the flow (fig. 3.2). Links are represented by curved arrows whose width are proportional to the square root of the flow. Higher flow nodes and links are darker than those with low flow. Undirected links are displayed as a curve without arrow head. The square root scale is used because there is often a long tail in the flow distribution which makes relative size hard to compare.
Once the layout is initiated, we start a simulation with the 20 highest-ranking nodes and run it until the forces have decreased enough (section 2.2.2), then we display it. When the layout occupies little screen space it is hard to see the most important features. To highlight those features, the layout needs to be filtered.

3.1 Level of detail

When you zoom in on a city on a map, at some point, the extent of the city is less important than the features of the city. The location of the main roads and landmarks is most useful while displaying the names of smaller streets or even addresses would clutter the map. If you look for something specific, you type the street name in a search box or zoom in ever further to where you expect to find it.

Taking this concept to visualization of hierarchical networks, the analogue is to show the highest ranking nodes in a module, the links between them with the highest flow, the names of a select few high ranking nodes, and continuously show more detail as the layout is zoomed (fig. 3.3).

This approach is called rank-based level-of-detail and can be thought of as rank-thresholding as a function of scale. The result is that only the most important features of a network are visible when the occupied screen space is very small.

3.2 Revealing module structure

When zooming in on a module far enough so that it could contain a low-detail network, its contents are revealed (fig. 3.4).
Figure 3.4: When a module is given enough space on the screen, its contained nodes or sub-modules are revealed. If the module contains sub-modules, the same process can be repeated until reaching the nodes on the lowest clustering level. Here, the content of the module named “Life Sciences” in fig. 3.3 is revealed.

Upon zooming further to the point that the module fills the screen, its background color is faded to white and its radius is increased. The result is that the module gets out of the way when it is so big that it no longer provides context to the outside world. Zooming in on modules within modules reveal their contents and so on.

The entire process works continuously. With each zoom change, check all visible modules if their size is big enough. For modules that are big enough, a new layout is initiated with the local origin translated to the center of the parent module.

3.3 User interaction

The entry point is the file dialog (fig. 3.5) where a clustered network file can be loaded by clicking “Load network”. For demonstration purposes, the application ships with clustered citation data, which can be accessed by clicking “Load citation data”.

The user interaction model is inspired by Google Maps. To zoom, you either use the scroll wheel on a mouse, or use a two-finger scroll gesture on the trackpad on a laptop. Clicking and dragging pans the view, and nodes can be moved around by clicking and dragging them. Clicking on nodes or modules highlights them with a red border (fig. 3.6) and displays their properties (fig. 3.7). Pointing at a node or module highlights incoming and outgoing links.
When a network is loaded, the layout is revealed together with the sidebar, which contains several different items (fig. 3.7). Starting from the top, a button which closes the sidebar (a) is followed by a field which displays the current filename (b).

Below the filename is a search box (c). Typing part of a node name or a regular expression matching node names lists the first 15 hits, and the modules containing the hits are overlaid with red circles whose area are proportional to the number of hits (fig. 3.8). This is useful for finding which module a certain node ended up in.

The item named occurrences (d) is used for loading lists of nodes. Several lists can be loaded at a time, and each is mapped to an unique color. The modules are then overlaid with circles whose area are proportional to the number of occurrences in that module. If there are nodes from more than one list present in a module, the one with the largest number “wins” (fig. 3.9). This is useful for investigating in which modules sets of data points can be found.

The selected module table (e) shows some metrics and statistics, including the editable node or module name, the path in the hierarchy, the contained and exiting flow, degree, the number of nodes, links and total number of children. The graphs below the table (f) show flow and degree distributions in linear and log scale.

Figure 3.6: The selected module “Life Sciences” which information is visible in fig. 3.7 (e-f).

<table>
<thead>
<tr>
<th>File</th>
<th>Lines</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>No files loaded</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add file

- Occurrences

<table>
<thead>
<tr>
<th>File</th>
<th>Lines</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>No files loaded</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Selected module

<table>
<thead>
<tr>
<th>Name</th>
<th>Life sciences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree path</td>
<td>1</td>
</tr>
<tr>
<td>Flow</td>
<td>0.0681</td>
</tr>
<tr>
<td>Exit flow</td>
<td>0.02872</td>
</tr>
<tr>
<td>In-degree</td>
<td>8</td>
</tr>
<tr>
<td>Out-degree</td>
<td>6</td>
</tr>
<tr>
<td>Nodes</td>
<td>256</td>
</tr>
<tr>
<td>Links</td>
<td>14196</td>
</tr>
<tr>
<td>Children</td>
<td>7533</td>
</tr>
</tbody>
</table>

- Module flow distribution

- In-degree distribution

- Out-degree distribution

Figure 3.7: The sidebar menu shows statistics from the selected module highlighted in red (fig. 3.6). From top to bottom, the menu shows the current filename, search box, point occurrence files, a table with selected module/node information, and graphs for module distributions.
Figure 3.8: (Left) Typing a query in the search box lists matches with their module assignments and highlights the modules with a circle which area are proportional to the number of hits. (Middle) Zooming in on a module with search matches highlights where in the module the matches can be found. (Right) On node level, search matches highlights the entire nodes.

Figure 3.9: Loading files with lists of nodes (occurrences) highlights in which modules the nodes are located with circles whose area are proportional to the fraction of nodes in the list to the total number of nodes in that module. The list of nodes which occur most often in a module “wins”. Here, three randomly generated lists are loaded and marked with blue, orange and green circles.
4 Results

The name of the application is Infomap Network Navigator and it is available at navigator.mapequation.org with complete source code available on the source code hosting platform GitHub.\(^1\)

In fig. 4.1, we can see how a typical session looks like. The header has been added to make the appearance similar to other projects by the team behind Infomap and the map equation, for example Infomap Bioregions.\(^2\)

Figure 4.1: Infomap Network Navigator running with the citation data loaded and the contents of several modules revealed. The selected module also has its in- and out-links highlighted and information about it is visible in the table in the sidebar. Below the table, the flow distribution graph is visible in log scale.

\(^1\) www.github.com/mapequation/network-navigator

extreme values for flow, while HNN uses the module’s extreme values. Because many modules are visible at the same time, the link colors and sizes need to be comparable between modules.

Figure 4.2: Hierarchical Network Navigator (left) and Infomap Network Navigator (right) zoomed in on the “Life Sciences” module. Both display 20 modules but different number of links. HNN uses link thresholding which means the amount of information visible is configurable, while INN uses level of detail for links which mean that link visibility depends on the level of zoom.

Infomap Network Navigator shows links both between nodes inside the module in the center of the screen, and to other modules outside the screen.
Conclusions

We described the implementation of Infomap Network Navigator, which is an application written for the web, with purpose of visualizing community structure in complex networks.

The advantages of the application are that it is written in JavaScript, and that it implements the idea about looking at networks as maps with a progressively increasing level of detail. It provides a platform for implementing features, both those present in the Hierarchical Network Navigator, but also new ideas such as the occurrences view and distribution plots.

Infomap Network Navigator lays the foundation for modernizing how users of Infomap interact with and explore their data, and serves as a tool for exploration and understanding of complex systems in a fast, intuitive way.

5.1 Future improvements

The map file format, which is supported by Infomap, was developed with visualization in mind but supports only two-level solutions. Generalizing this format for multi-level networks is an obvious next step. The ftree format does not support module names which map does.

The Hierarchical Network Navigator (HNN) has several features which are missing in Infomap Network Navigator. Binary file format support for streaming only needed parts of the network would speed up loading times and increase the file size limit.

In HNN, you can change the color and size range for nodes and links, and choose between linear, root and log scales (fig. 5.1). Infomap Network Navigator only uses root scale. More control over node and link filtering is missing and would be a nice addition.

Exporting the layout to PDF would be convenient for use in articles, and exporting graph data would be useful for analysis in other programs, such as Matlab or Excel.

Finally, using a technology called Canvas for rendering could improve performance.
6

Acknowledgements

First, I would like to thank Martin Rosvall for his vision, ideas, and letting me be a part of the group at IceLab these past 6 months. Daniel Edler helped me push myself further when everything seemed impossible, which had a significant impact on the result. His prior work set the benchmark for what I wanted to achieve with my work. Christopher Blöcker listened to my questions about data structures and software patterns, made me appreciate functional programming languages, and proof-read this thesis thoroughly. Ludvig Bohlin let me bother him with questions in the middle of his thesis writing and always made me feel welcome. He also provided insightful comments which improved this thesis. Alexander Ramström beta tested my work and gave valuable feedback.

I would also like to thank all people at IceLab for their welcoming and support.

This work is based on the unpaid work of many free software authors, for which I am grateful.

The citation data shipped with this application is courtesy of and authored by the Bibliometric group at KTH.
Bibliography


