Network Visualization Visual Editor

Xiaoyang Chen

Master of Science

School of Informatics

University of Edinburgh

2023

Abstract

Previous work on network visualization has mainly focused on node-link diagrams and is usually based on programming, which limits expressiveness and accessibility. In this project, I propose to explore ways to design expressive network visualizations using low-threshold methods. I design and develop a user-friendly graphic interface based on NetPanorama, a grammar that supports flexible visualization solutions for network data, allowing users to create network visualizations without programming. Users can explore different visualization solutions and seamlessly interleave data binding and visual designing. I then conduct user tests on the interface. The result shows that participants can swiftly learn how to use it and design various visualizations.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: 329009
Date when approval was obtained: 2023-08-02
The participants' information sheet and a consent form are included in the appendix.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Xiaoyang Chen*)

Acknowledgements

I would like to thank my Supervisor, Dr. Benjamin Bach, for his guidance on this project. I would also like to thank Xinhuan Shu and Jinrui Wang for helping me a lot with understanding NetPanorama specification. Then, I would like to thank all the volunteers who participated in the user study. Finally, many thanks to my friend Xichen Liu for making the days of writing my dissertation not so boring.

Table of Contents

Chapter 1

Introduction

Network visualization aims at visually presenting interconnected entities and the relationships between them. An exemplary network visualization graph is a node-link diagram in which nodes represent identities; links represent their relationships, and visual attributes like color, shape, position, etc., can be used to encode data domains. For many years, researchers and data scientists have developed numerous graphs beyond node-link diagrams for visualizing network data, including adjacency matrices, adjacency lists, chord diagrams, small multiples, etc.

Like any other visualization designing process, network visualizations require iteratively exploring, testing, and evaluating solutions on given data, tasks, and users [29]. Many tools and programming libraries are available that can create network visualizations. However, most of these solutions are not designed specifically for network visualizations, and they provide few supports besides node-link diagrams.

NetPanorama is a declarative grammar currently being developed by VisHub at the University of Edinburgh for visualizing network data. It provides a high-level specification for network visualization design, which supports various kinds of network visualizations, and allows users to customize them deeply. Moreover, it has excellent support for multivariate, temporal, and geographic networks. However, writing declarative specifications requires programming expertise, and learning the grammar itself has a steep learning curve, hence not suitable for the general public.

This project proposes a visual editor for manipulating NetPanorama. The goal is to foster the use of NetPanorama and allow users to explore different solutions and design network visualizations without programming swiftly. Meanwhile, the editor is designed to be straightforward with the critical abilities of NetPanorama emphasized, which is for targeting non-expert users and reducing the difficulty of implementation. As a result,

the interface enables fast exploration of different visualization layouts and focuses on mapping data to visual attributes to enhance expressiveness. I made prototypes and an interactive implementation for the interface. Then a user study was conducted, which showed that users could learn the interface swiftly and create rich variations of visualizations, as well as revealed the existing problems.

The work will be integrated into **Vistorian**, a free, open-source online tool for network visualization, and will be released later this year to the public. Vistorian currently supports constructing networks and creating pre-defined visualizations based on NetPanorama templates. Users can search for nodes with specific labels and filter links based on types, but they can make minimal changes to the templates. This project will allow users to design and customize visualizations in Vistorian, and a demo of the work is available at https://vimeo.com/857787034

Chapter 2

Background and Related Work

2.1 Network Visualization

Sketching and vector design editors are two traditional but popular ways of designing visualizations, including network visualizations. These methods are easy to implement with few limitations but lack computational thinking and data manipulation [26].

Imperative or declarative programming. Many tools are built for universal visualization purposes, but they provide reasonable abstractions for designing network visualizations. Protovis [6] is an extensible toolkit for constructing visualizations by composing simple graphical primitives, which reduces the learning threshold and maintains a high level of expressiveness. D3 [7] directly maps data attributes to elements in the DOM (document object model) with a visualization kernel. Processing [21] is a programming environment built for media art, focusing on developing visually oriented applications with an emphasis on interaction and animation. Vega [28] also has excellent interactivity by supporting real-time updates, while Vega-Lite [27] reduces some expressiveness but simplifies its specification writing. In addition, Python and R languages have numerous libraries that support network visualization, especially node-link diagrams, such as NetworkX [12], PyVis [20], Plotly [31] in Python and ggraph [30], igraph [9], visNetwork [2], NetworkD3 [1] in R. Cytoscape.js [10], Sigma.js [8] and G6 [34] are some representatives of web-based libraries, which have powerful expressiveness but are still limited to node-link diagrams. NetPanorama, on which this project is based, focuses on visualizing network data and supports design far beyond node-link diagrams.

Graphical interface. General purposes visualization tools like Lyra [25], Data Illustrator [18], and Charticulator [23] have expressiveness comparable to programming-

based tools, but they have no direct solution for network visualizations. Gephi [3] provides easy and broad access to network data and allows for spatializing, filtering, navigating, manipulating, and clustering while providing rich layouts and visual design. Graphies [24] provides a streamlined workflow that allows users to create expressive designs effectively, and it goes beyond basic undo to make users explore multiple solutions by allowing different branches to exist simultaneously.

2.2 Visualization interface design

According to Grammel et al. [11], the template editor and the visual builder are optimal solutions for visualization designing interfaces. Template editors such as Tableau [13] and ManyEyes [33] enable a swift, top-down process of design visualizations, but they restrict users' ability to customize their diagrams. Visual builders like iVisDesigner [22] usually follow a bottom-up workflow, which allows users to map data to visual elements and combine them together to make complex visualizations. Mendez et al. [19] find that visual builders are less efficient than template editors, but they encourage users to explore different visualization designs.

In recent years, research on visualization interface design has focused on improving user experience and expressiveness. Lyra [25] makes visualization design accessible to a broader audience by reducing both tedium and required technical expertise. It provides direct manipulation techniques such as handles, connectors, and drop zones to ease execution and visual canvas to improve evaluation ability. Data Illustrator [18] can bind data with shape primitives and layouts. It applies a logic called lazy data binding, which only allows binding data to visual attributes when necessary. When users click the binding icon near a property, a list of applicable data columns appears. Charticulator [23] also follows the procedure of building visualizations with glyph primitives, and it allows the construction of compound glyphs. Users are able to create bespoke and reusable layouts rather than choose from templates. Despite its excellency in general visualizations, concepts that are common in network visualizations, like nodes and links, are not explicitly declared, and force-directed layouts are not supported. Graphies [24] allows customization and manipulation of individual elements. Moreover, it introduces a timeline feature that enables users to trace back to any state of the diagram on any branch. It also enables smooth, animated transitions between different visualizations, which can support a rudimentary form of storytelling.

Chapter 3

Design Process

This section details how I plan, design, and implement the editor interface for Vistorian. First of all, a thorough exploration and analysis were conducted to evaluate which features can be possibly integrated into the interface and how they may contribute to the design process. This was followed by an iteration of the prototype design. Finally, I implemented a usable web interface for testing significant features in the prototype.

3.1 Conceptualization

Vistorian renders graphs with NetPanorama. The current version of Vistorian enables users to upload data and construct networks, and it provides several visualization templates to render the networks. However, users can barely modify the look of the graph and cannot change how data are bound to visual opponents. Therefore, the expressiveness of NetPanorama is significantly restricted. An optimal interface should allow users to profoundly modify NetPanorama specifications without programming. In other words, they work on another level of abstraction and do not need to understand how NetPanorama works.

On the one hand, NetPanorama's powerful visualization capabilities provide great potential for an interface to manipulate them; on the other hand, its complexity also brings significant challenges. Considering the complexity of the task and the short duration of the master project cycle, I have streamlined the workload based on two strategies to ensure the project can be completed within the timeframe while contributing to the proposed goal:

- The target users consist of non-professionals who may need to include network visualization in their workflow, especially those without programming skills. So I

discard the features in NetPanorama that rely on professional knowledge, require complex configuration, or have the least effect on the graphs. This approach aligns the project outcome with reasonable expectations and allows users to swiftly get started and produce rich visualization designs.

- The extensibility of the editor was emphasized during the design phase so that subsequent development could be built on what has been achieved. Moreover, the features are coded in order from easy to difficult at the implementation stage.

I conducted a detailed study on NetPanorama to understand its specification and evaluate which features can be added to the prototype. Notably, NetPanorama is still under development, and this project is based on its existing functionality.

The specification of NetPanorama is written in JSON form. It describes network visualizations through a pipeline of steps, each corresponding to one or more grammar structures in the JSON specification. Primitives are defined in these structures to express designs. The pipeline and the schematic diagram of each step are shown in *Figure 3.1*. Most of the steps and structures are optional, and they can appear in any order in the specification regardless of the conceptual order, which guarantees much freedom in terms of the design and implementation of the visual editor. Although some steps may refer named objects from other steps, they should be viewed as independent components rather than a form of subordinate relationship. For example, users can independently adjust the layout and node appearance without affecting each other.

Importing data and **Constructing networks** are two essential parts of the early stage of network visualizations in Vistorian. Users finish these procedures by following several instructions, from uploading data files to extracting information about nodes and links. **Transforming data tables** is helpful for modifying data through filtering and aggregation but should also be considered as pre-processing before the actual designing of visualizations. Hence, these procedures are not considered for the editor interface.

Transforming a network allows users to make specific calculations and modifications on the network, which acts on nodes and edges and can change the network's topology. The transformations include aggregation or filtering of nodes or links, projection of edges, etc. This step has excellent freedom and potential, depending on what narratives users want. Nevertheless, conducting network transformation for specific data usually involves writing reasonable expressions, no less than having the users fill in the specification. Moreover, although network transformation does not act directly on the data table, it works similarly to pre-processing the data before constructing networks.

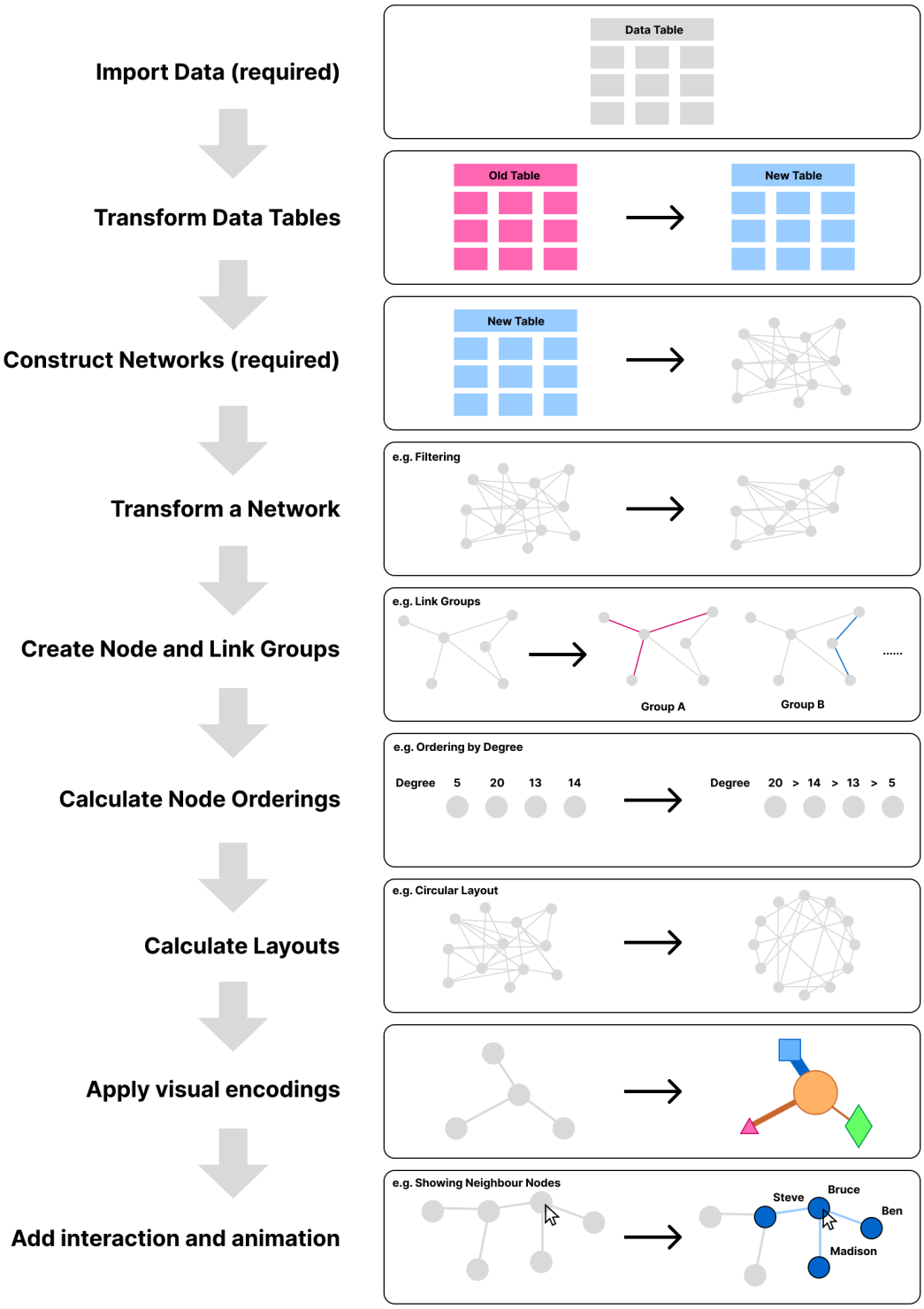


Figure 3.1: The pipeline of NetPanorama, with each step containing a schematic diagram. Most of the steps are optional and can appear in any order in the specification.

The original intent of this visual editor is to make it straightforward for users to quickly explore different visualization ideas rather than waste their time managing data. Also, Vistorian provides an easy and intuitive way to manipulate data by allowing users to filter data on certain domains after creating visualizations. However, NetPanorama provides the calculation of node metrics, e.g., degree and closeness, based on network structures. These metrics are helpful for expressiveness because they usually reflect on some characteristic of the identities in a particular narrative background (e.g., nodes with higher degrees are usually more important) and can be calculated in advance for users to access without extra work.

Three steps significantly affect how elements (e.g., nodes and links) are positioned and arranged on a graph. **Groupings** divide data of nodes and links into multiple sets based on common attributes, which is helpful for creating small multiples, with each group rendered separately. **Orderings** can be applied to sequentially position nodes in specific layouts, e.g., linear layouts and matrices. **Layouts** have the most prominent effects on visualizations and are gigantic in quantity, which helps create diverse visualizations. NetPanorama currently divides layouts into three major categories:

- **Procedural layouts**, e.g., force-directed layouts and constraint-based layouts, determine node coordinates from their relevant positions. These kinds of layouts are usually calculated by advanced algorithms with adjustable parameters. Users without relevant expertise may find it difficult to understand these mechanisms, and the results from specific settings can be unpredictable.

- **Geometric layouts** arrange elements based on geometric structures, e.g., linear, circular, and cartesian (coordinate system) layouts. Most of these layouts arrange nodes according to their attributes. For example, users can arrange the x-coordinates of the nodes in the coordinate system according to their degree and the y-coordinates according to their dates.

- **Table layouts** conveniently group data based on two fields. It is typically used for creating adjacency matrices, in which horizontal and vertical labels usually represent entities and matrix elements usually represent relationships between entities.

Visual encodings are necessary for all visualizations, and they determine how elements look on a graph. Currently, NetPanorama enables setting element attributes such as position, color, shape, and size. These attributes can be set as constant values or

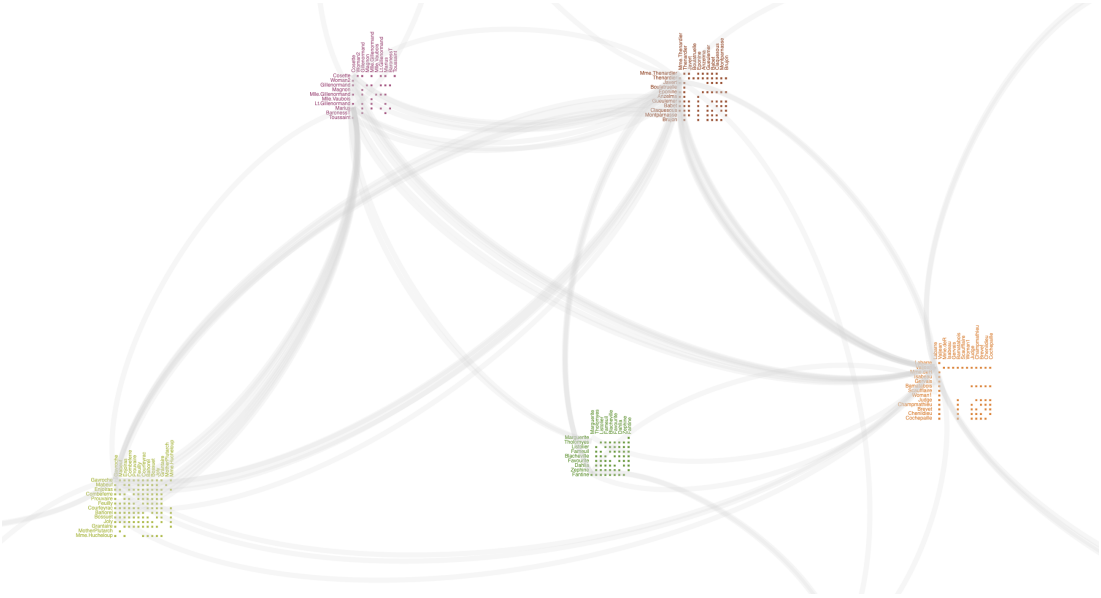


Figure 3.2: An example of complex layouts in NetPanorama by the nesting of visual encodings

calculated by conditions and expressions. Moreover, they can be bound to certain data domains by setting up scales, which is helpful for storytelling. For example, one can bind the size of nodes with their degree so that nodes with larger degrees appear to be more prominent on the diagram, or bind a color scheme with link types so that different types of links look different in color. Nesting of visual encodings is also possible. Instead of defining simple visual marks, a visual encoding structure in NetPanorama can include other structures, such as groupings, or even another visual encoding structure, which helps to make sophisticated layouts. For example, the visualization shown in *Figure 3.2* is created by firstly grouping nodes based on clustering, with each group placed under a force-directed procedural layout. Then nodes and links inside each group are rendered as a matrix. Finally, links across different groups are rendered as curves. Further nesting can still be possibly accomplished. For example, cells in the matrices can be rendered as more complex glyphs or subgraphs.

NetPanorama also supports **interaction**. It is possible to add tooltips, drags to reposition nodes, zoom, and pan screen. However, dragging nodes can only change the position of the current node, and it works by re-rendering the node without smooth animations or dynamic rewriting that can be found in some tools like Vega. Also, by setting parameters in the specification, values can be bound to input components, e.g., sliders or checkboxes, which enables simple adjustments on the diagram. Vistorian has implemented some simple examples with this feature, such as a slider that can set

the size of all nodes to one value. However, these built-in, experimental components have limited styles and can only realize simple modifications to individual values, thus not suitable for the visual editor proposed in this project. Another noteworthy interaction is selection, which can be used to update the visual appearance of selected elements or derive new selections (e.g., select neighbor nodes of the selected nodes, or show the shortest path between two selected nodes). A selection can be defined with a type (e.g., cursor, lasso), an event (e.g., click, mouseover), and a key (e.g., "alt", "ctrl") and can be bound to simple actions like change selected nodes into another color declared in the template. These interactions are mostly basic and can do minimal, pre-defined changes to the design. They are not built to support complex data binding logic or deep customization on a single element. However, they can be used while viewing the created visualization (e.g., highlighting the selected elements). Besides, it is possible to make **animations** between successive values of a parameter, which can be used as a way to demonstrate diagrams but does not help in designing visualizations. Notably, NetPanorama currently re-renders for each new parameter frame, which means interaction or animation are not smooth.

A complex design like *Figure 3.2* may contain multiple, nested use of groupings, orderings, layouts, and visual encodings, which is difficult for non-professional users to manipulate, as well as challenging for implementation. Also, enabling users to customize these steps entirely is easy to generate illegal specifications (which cannot be rendered) or meaningless diagrams, resulting in frustration or confusion. To make the user experience more straightforward, the solution in this project is giving up some expressiveness by not letting users follow the logic of NetPanorama. Instead, the solution is similar to a template editor. Visualizations are categorized into several major layouts for users to choose from, each rendered on a pre-defined template. Users design visualizations based on templates rather than starting from nowhere.

Typical template editors restrict users' freedom of creativity. However, I want users not to be limited by templates but to be motivated to explore various designs. To accomplish this goal, firstly, the major layouts are classified with a different criterion from NetPanorama, avoiding excessive refinement of categories, at the same time, considering common categorization methods and the ease of implementation. Users should be able to create vast designs within the chosen layout. Second, users can choose multiple layouts at a time, and they can design and compare them in one workspace. Third, users should be able to deeply customize visual attributes to enhance expressiveness. Numerous studies [32][4][25][17][18][5][16] have observed that typical

network visualization construction of expert designers involves two types of tools. One is a graph analysis tool like Gephi for creating subgraphs and binding data to visual attributes; the other is a vector design tool like Adobe Illustrator for manually adjusting visual design. This workflow is inconvenient and, more importantly, creates an artificial boundary between two steps. An obvious flaw is that when designers switch from graph analysis tools to vector design tools, they lose information about data. If they want to change the visual variables related to data in the first tool after customizing the design in the second tool, they must redo everything again. This inconvenience inspires this project to provide users with various data binding choices and visual design solutions. More importantly, data binding and visual designing should be seamlessly integrated into a single workspace, and users can interleave them without conflict and get instant feedback.

In specific, users follow the following steps:

- Import data and construct networks
- Choose one or more from several major layouts, including a node-link diagram, a matrix-based diagram, a diagram resulting from groupings and nestings (e.g., small multiples), a map-based diagram, etc.
- Design visualizations from the base layout(s) in the visual editor interface, which can include further changes on the layout(s), data binding, visual designing, and switching the visualization currently being worked on.

The first two steps have been implemented in Vistorian. Users are able to upload data and construct networks by specifying the data columns that correspond to the node and link attributes. The work done in this essay contributes to the third step.

Based on previous analysis on NetPanorama and other studies on design workflow, The interface is expected to focus on supporting diverse layout solutions and making it easy for users to change visual attributes of elements by directly manipulating them or binding data (including metrics from data) with them. Groupings and Orderings are supported, but only corresponding options appear based on their chosen layouts. In addition, simple interactions can be done by selection, but animations are not considered. Transformations, especially filtering of data, can help design visualizations. Nevertheless, users can conduct filtering while exploring the visualization they create, and supporting them at the design stage is unnecessary since being simple is suitable for demonstration purposes.

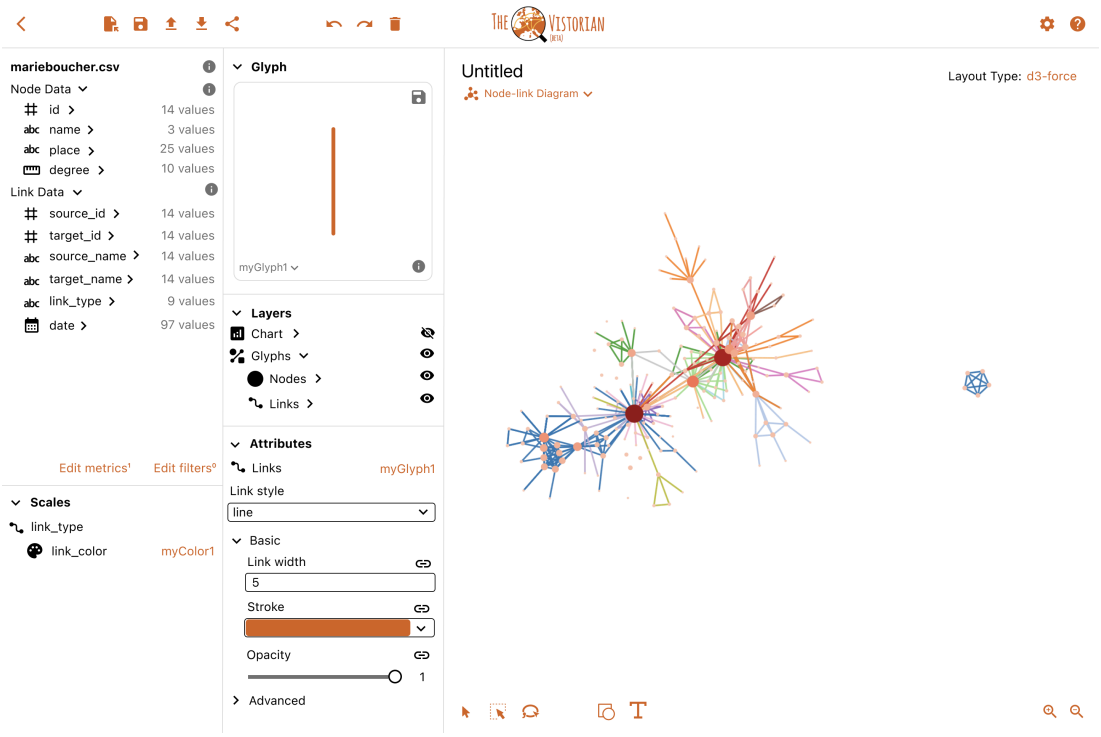


Figure 3.3: The earlier version of the prototype.

3.2 Prototyping

There are two major versions of the prototype being made. The earlier version of the prototype is shown in *Figure 3.3*, which features a relatively professional style and includes some ideas that go beyond NetPanorama as a concept for the future. The prototype consists of three parts: the toolbar at the top, the design panels on the left, and the visualization display on the right.

The toolbar has some general and necessary features, such as undo, redo, save, share, delete, settings, and help.

The design panels are responsible for different functions. On the top left is a data preview on nodes and links; users can also include node metrics. Besides, they can also apply simple filtering to the data. On the bottom left, there is a scale panel that shows information on data binding. Also, users can directly change visual variables and binding relationships. Then there is a glyph panel that shows a preview of visual elements. Thanks to the declarative syntax of NetPanorama, users can also save and reuse the styles they create. The layer panel demonstrates the different elements in the visualization and their hierarchical relationship. Finally, an attribute panel allows users to conduct visual design. They can either set an attribute to a constant value or bind it

to a specific data column simply by clicking the icon on its right.

The visualization is displayed on the right of the interface. Here users can change the category of the visualization and adjust its layout. Also, they can easily zoom in, zoom out or move the diagram. In addition, on the bottom left of the visualization. I envision a future possibility where users can choose from different selection methods and adjust the attributes of the selected elements. Besides, they can add shapes or texts to the diagram to create titles, legends, tooltips, etc. NetPanorama currently only supports full adjustment on global characteristics and cannot create customized glyphs like vector design tools.

This prototype was abandoned because it is too complicated for a quick experience and too ambitious for implementation. After group discussion, I finally created and adopted the second prototype, as shown in *Figure 3.4*. The design and implementation center on node-link diagrams. However, they are designed to be easily extensible to other network visualization types. This sets a reasonable goal for the dissertation, as well as the path for future work. In addition, I choose node-link diagrams because they represent the most common network visualization genre and produce various layouts easily, which is suitable for demonstration.

In this prototype, the toolbar is preserved. The panel has two modes, the exploration mode and the design mode. The exploration mode is made for users to view and explore the visualizations they create, and it will integrate some existing functions in Vistorian, like searching and filtering. This project focuses on prototyping and implementing the design mode, where the panel is divided into four sections:

- General settings, where users can adjust attributes related to layout. Further settings related to a specific layout appear only when that layout is selected. This design philosophy of presenting only the necessary information permeates the entire user interface to maintain simplicity. For example, if users choose the Cartesian layout, they can bind node horizontal and vertical coordinates to node attributes. All features that are not part of the node and link appearance are also placed in this section. For example, the value of label importance is used for displaying labels of nodes with a degree higher than it.

- Node appearance, where users can specify the nodes' look. Here I show some possible patterns with different settings, including shape, opacity, visibility, size, and color. Generally, a setting consists of two parts, the value (constant, range, scheme, etc.) on the left and the bound data on the right. The "Data Binding" part

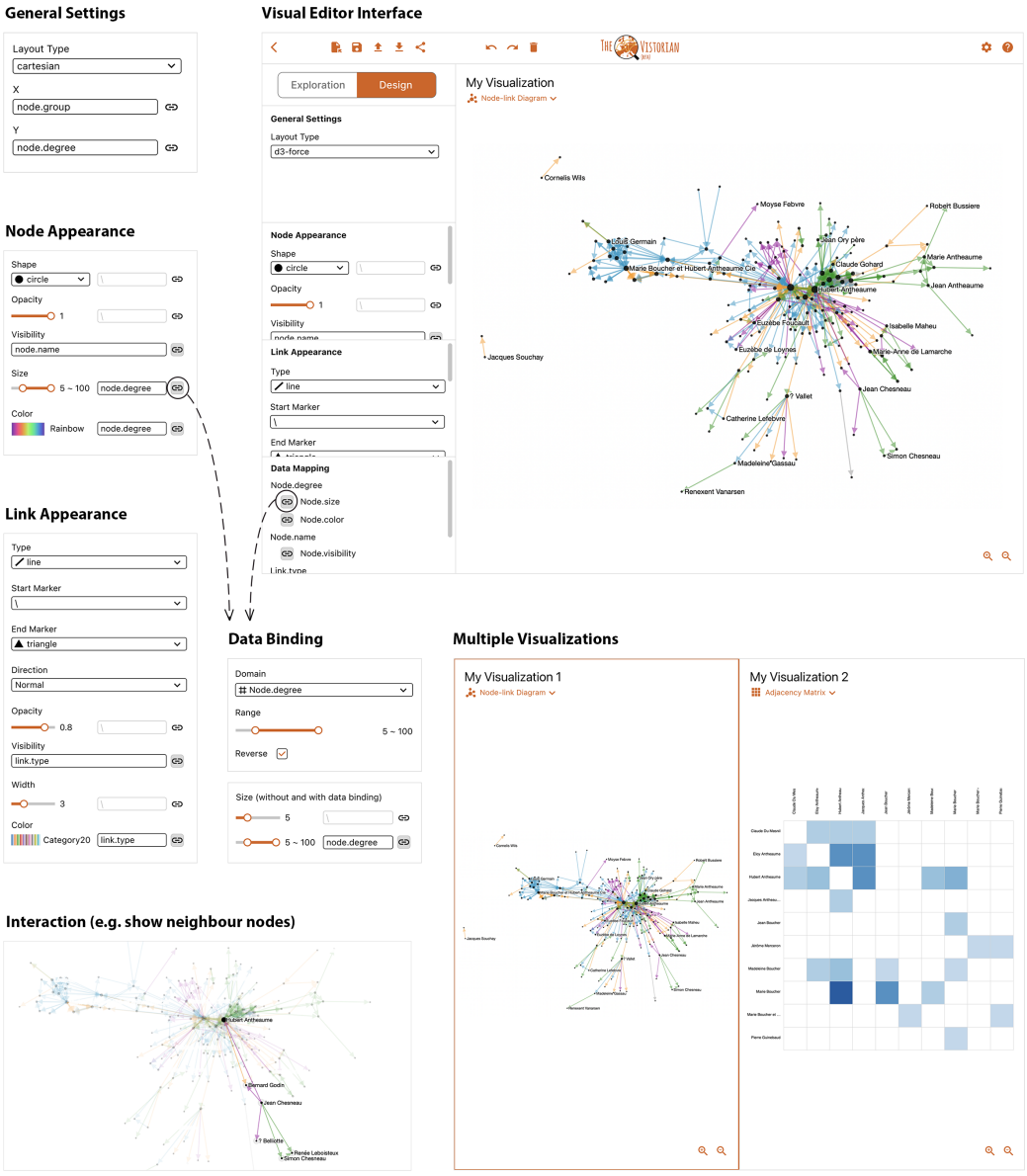


Figure 3.4: The second version of the prototype

in *Figure 3.4* shows how this mechanism works. When users click the ”chain” button, a small window appears where they can specify what data to bind, what value range or scheme to use, and other settings, like if they want the mapping to be reversed. Also, generally, there are two different states of every bindable setting with the example of size setting. When size is not bound to any data column, users can set it to a single value with a slider; when it is bound to a data domain, the slider becomes a range slider so that they can specify the maximum and minimum value of node size mapping to degree. The visibility setting shows an exception, where users only need to specify what data column to bound, and they can decide on nodes with specific values to be visible in the pop-up window. Notably, data binding can be tricky sometimes because many combinations are meaningless from a narrative perspective, but users are not prevented from using them as long as they are legal. However, the data column type (categorical or numeric) is determined automatically to reduce unintelligible binding options.

- Link appearance, which works similarly to the node appearance section. Examples include type, start marker, end marker, direction, opacity, visibility, width, and color, some of which are not bindable. Both node and link appearance sections can be easily extended to visualizations other than node-link diagrams, as they all have the semantics of nodes and links. For example, in an adjacency matrix, row and column labels represent nodes, while cells represent relationships between nodes.

- Data Binding shows how data is bound to visual attributes in real-time. This section aims at emphasizing the importance of narrative through data. Users can intuitively see data binding relationships and adjust them here.

The adjustable settings shown in the prototype serve as examples. In the real scenario, node and link attributes can be much more complex. For example, the color of an element is usually defined by stroke (edge color) and fill (interior color). Also, attributes like the size and color of the start marker, end marker, and link path can be set separately.

Users can design and compare multiple visualizations in one workspace by selecting more than one solution after constructing networks, but they can only work on one visualization at a time. Clicking the area of a visualization highlights its frame and switches to the corresponding design panel. Users can zoom and pan the current visualization, and they can also switch to other types. In addition, the interface is

expected to allow users to interact with the visualization. When a user hovers the mouse over a node, its information, such as the label, is displayed, and its neighboring nodes are highlighted.

3.3 Implementation

I implement the prototype, mainly focusing on the design panel, for testing purposes based on React.js, as shown in *Figure 3.5*. The implementation follows the prototype's design, but certain adjustments are made. The main differences are:

- The ”chain” button is removed, which is supposed to trigger a pop-up window for setting detailed options associated with data binding. Currently, the options include data range, column to be bound, and whether to reverse data mapping. Before new options are introduced, using a pop-up window or secondary menu is unnecessary. Instead, the solution is that when the user binds data with a visual attribute, a small ”reverse” icon appears on the right of the attribute name, and the user can click it to reverse data mapping.
- Also, the data binding section only displays information about data binding but cannot edit it.
- Tooltips are added for some settings to help users understand them.
- The implementation uses components provided by Ant Design, a user interface library for React.js, with different visual styles from the prototype.

In the general settings section, I implement some representative layouts, including two procedural layouts, one is a force-directed ”d3-force” layout, and the other is a constraint-based ”WebCoLa” layout; and four geometric layouts, which are circular, linear, cartesian and polar layouts. Users can adjust basic parameters for procedural layouts and set ordering methods or bind node positions with data for geometric ones. In addition, they can decide how many nodes' labels to display to keep the visualizations tidy.

In the node appearance section, users can customize nodes' size, shape, opacity, fill, and stroke. Here, I provide users with plenty of color schemes, such as categorical schemes for classification, single-hue schemes for displaying sequential or numeric data, and diverging schemes for comparing two extremes.

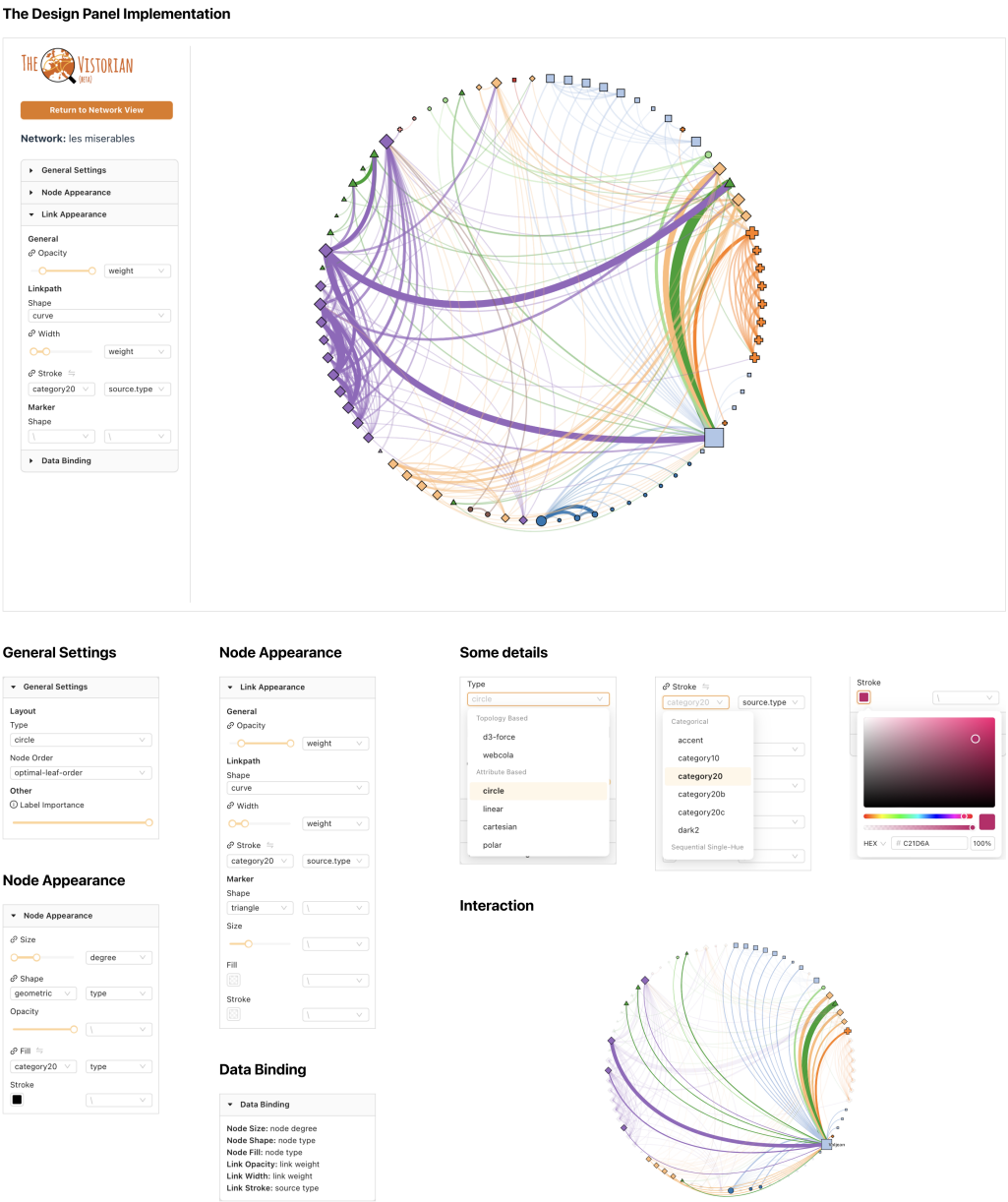


Figure 3.5: The Implementation of the prototype

In the link appearance section, users can customize links' width, shape, opacity, and stroke. In addition, they can add optional end markers for links, with size, shape, fill, and stroke being editable. Furthermore, they are also allowed to bind link attributes to not only link data, but also the data of their source or target nodes.

Users can seamlessly adjust every setting, and the diagrams re-render in real time to provide instant feedback. They can check how visual attributes are determined by data in the data binding section. Furthermore, users can zoom and pan visualizations, as well as display neighboring nodes by hovering the mouse pointer over any node. Last but not least, the initial style of the visualizations is set to be fixed and simple to prevent pre-defined, elaborate styles from influencing the users' choices.

Chapter 4

User Study

This section details how I conducted a first-use study to gather empirical observations about 1) whether users can learn how to use the visual editor swiftly and 2) how effective the interface is at helping users create various expressive visualizations.

4.1 Procedure

The user study is conducted separately with each participant. First, the users are introduced to network visualizations and Vistorian with presentation and demonstration using a dataset of commercial transactions. The tutorial (around 15 min) follows a script, including uploading data, constructing networks, choosing visualization types, and designing the visualizations in the editor interface. Each time a new feature is introduced, users are free to try it themselves. After the introduction, they are asked to reproduce the visualization shown in *Figure 4.1* based on the same dataset to ensure they understand how to use the interface. During the reproduction, they can see information about data binding in the target visualization and ask any questions.

Except for preparation work that happened before users enter the editor interface, to reproduce the visualization, they need to perform these steps (in any order) correctly:

- Choose a ”WebCoLa” layout.
- Turn off all node labels by setting label importance.
- Bind node size to degree and adjust the range.
- Bind node fill to degree and set the color scheme to ”grey”.
- Set link shape to ”squiggle”.

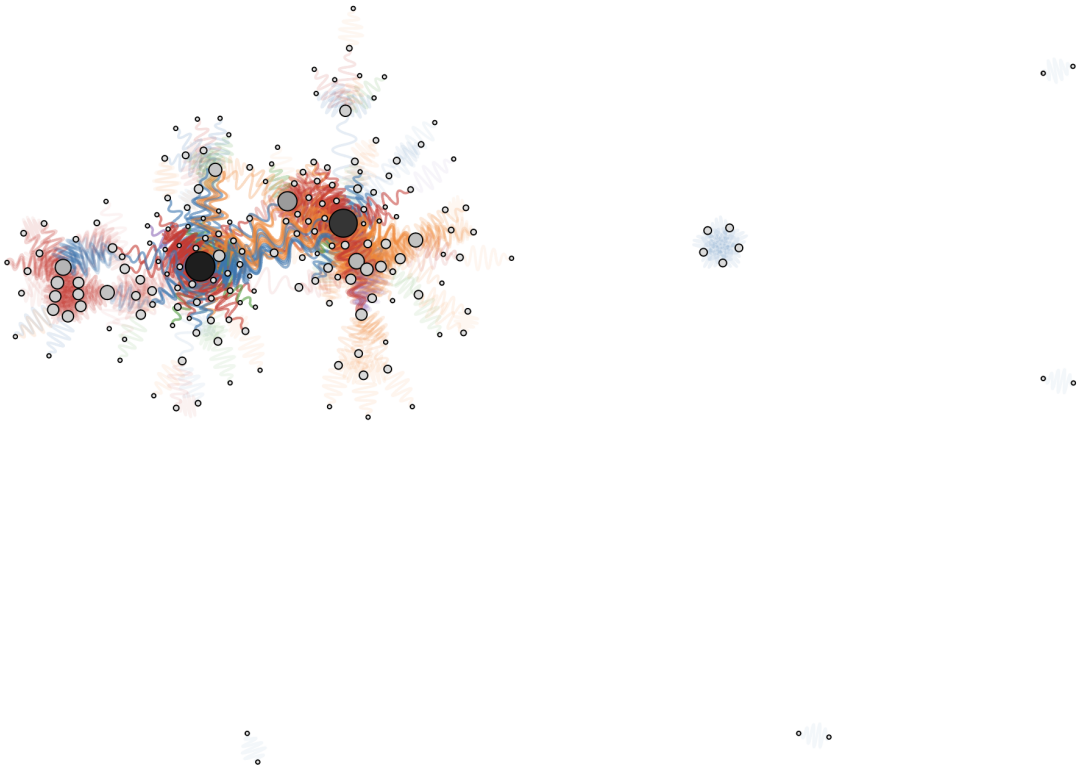


Figure 4.1: The visualization that users need to reproduce

- Bind link opacity to link source degree and adjust the range (this means links from higher degree nodes appear to be more opaque).
- Bind link stroke to link type and set the color scheme to ”category10”.

Participants then need to perform an open-ended task, where everyone is asked to create a visualization based on a dataset of characters from the French novel Les Misérables. The dataset contains a node table of characters, with each row being a character name, a unique number as ID, a group number he/she is assigned to, and a link table where each link represents the relationship between two characters. Every link has a weight, indicating the closeness of their relationship, as well as a direction from a source node to a target node. Participants started the task by uploading data, and they were asked to follow the think-aloud protocol [15], which means they should say whatever comes to their mind as they complete the task. Also, they can experiment with different solutions and design their visualization without a time limit until they think they are satisfied with the result.

After the user study, participants are asked to give some feedback on the editor, e.g., what they like/dislike about it and what features they want to add.

4.2 Participants and Apparatus

Five volunteers (three males and two females, aged 22 to 24 years old) participated in the study. All of them are post-graduate students at the University of Edinburgh, out of which one was from Edinburgh College of Art (P1 in *Figure 4.2*), one was from the School of Engineering (P4 in *Figure 4.2*), and three were from the School of Informatics (P2, P3, P5 in *Figure 4.2*). All participants had done some visualizations in their previous work, four of which had used programming libraries like Matplotlib [14] in Python, but only one had experience in network visualizations. The user study was conducted on a 13" M1 MacBook Pro, which was also used to record screen and voice.

4.3 Result

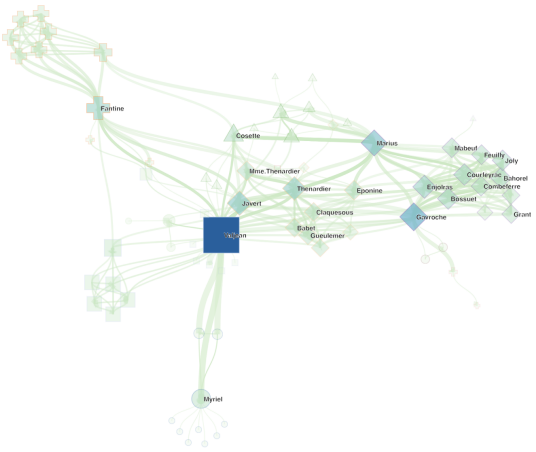
All participants successfully reproduced the target visualization within 5 minutes. Given the data-binding information, they had no problem understanding how this visualization is produced. However, two of them struggled when they tried to choose the correct color scheme, but they finally managed to do it. Also, participants did not know the exact values for some attributes, but they were able to adjust these values to make the visualization look almost identical to the target.

For the open-ended task, the average completion time was 321s, with a minimum of 178s and a maximum of 609s. *Figure 4.2* shows the visualizations produced by all five participants.

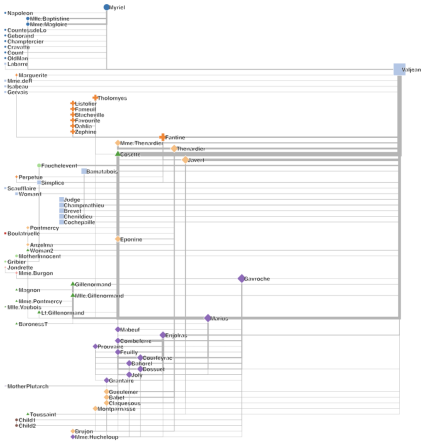
Although I did not ask participants to bind data to specific visual attributes, all of them did so as expected. Everyone bound node degree and node type with at least one visual attribute; four out of five bound link weight to at least one visual attribute. Participants took varying amounts of time to complete the task. However, each explored different layouts and visual options, two of them iterated back and forth to adjust the layout and visual attributes, and they all spent some time improving the expressive and aesthetic value of their diagrams. Furthermore, participants commented positively on the accessibility and expressiveness of the interface, although more layouts and visual attribute choices are wanted.

However, several problems were exposed during the user study:

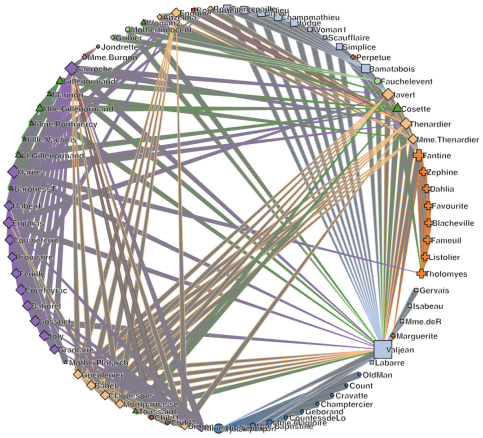
- None of the participants opened the data binding section when they designed their visualizations. In the implementation, users could only open one section each time, so they focused on the sections that could change the look of the



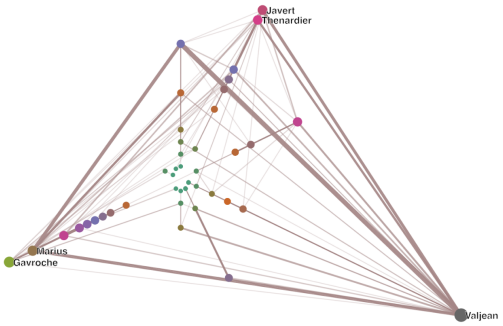
P1



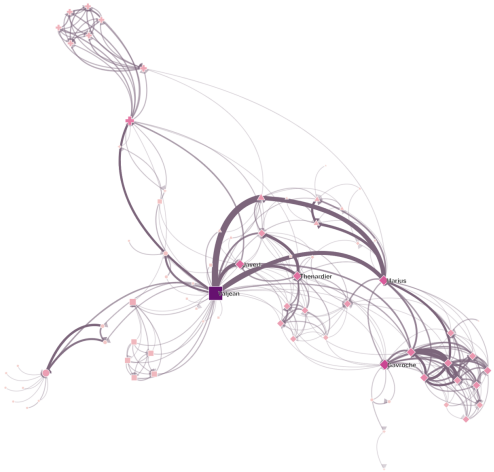
P2



P3



P4



P5

Figure 4.2: The visualizations created by participants

diagrams and ignored the data binding one. More importantly, all participants provided negative feedback on the current function of this section. Four out of five participants think the data binding information should be displayed near the visualizations rather than in a section in the design panel.

- Currently, the implementation does not indicate whether a data column is numeric or categoric, and it does not prevent users from binding data with visual attributes as long as they are legal. Generally, binding numeric data with numeric attributes or binding categoric data with categoric attributes are semantically reasonable. However, two participants (P1, P4) were observed to bind numeric data with categoric attributes or categoric data with numeric attributes without realizing it. Both of their situations happened when they tried to bind data with color scheme options, where only texts but no preview images of these schemes were shown. Specifically, participants 1 bound link type (categoric) with a single-hue scheme, while participants 4 bound node degree (numeric) with a categoric scheme.

- Two users tried to look back at the network data (not shown in the editor interface) and reported that they forgot the data structure.

- Three participants reported that they found some options with confusing professional terms even though they could see the effects by trying, especially with the node ordering methods (e.g., ”optimal-leaf-order”, ”barycenter”). Also, they thought the tooltips used for explaining some features were helpful and could be used to clarify these terms.

- Another problem occurred with the experimental setup itself. The datasets used in both tasks had limited data columns, which in turn limited the editor's ability to produce various visualizations (e.g., links in the commercial transaction dataset do not have weights, and links in the Les Misérables lack link types).

Participants also indicated some features that they expected to have in the future:

- All participants mentioned wanting information such as legends near the visualizations. Specifically, they mentioned two scenarios: one is automatically generated information to prevent them from getting lost in what was going on, and the other is the function of adding legends to enhance the readability of diagrams for viewers.

- Three participants wanted the ability to freely customize the visual attributes and positions of elements by directly manipulating them because they found the need to emphasize particular elements rather than modify all of them at the same time.
- Two participants thought an interactive tutorial was helpful for learning how to use the editor, one of whom stated that the process of constructing networks with instructions on each step was more friendly than the editor interface.
- Participants generally gave positive feedback on the feature of showing neighbor nodes. However, two thought there should be more advanced interactions in one workspace, e.g., showing detailed information on nodes and displaying relationships between two selected nodes.

Chapter 5

Conclusion and Future Work

In this project, I designed and implemented a visual editor interface for designing network visualizations with NetPanorama. Considering the project's time limit, the prototype and implementation focus on node-link diagrams with significant features in NetPanorama being emphasized, but the design can be extended to other types of network visualizations. The design panel allows users to perform data binding and visual designing in one workspace seamlessly. In addition, various visual mapping options and a flexible workflow enable fast exploration of different visualization solutions. The user study shows that the proposed design panel can be mastered by users swiftly, and it can nourish a rich set of network visualizations by encouraging exploration.

Nevertheless, there are still some limitations with the interface, which should be resolved gradually in the next stage of development. First, there should be a more vivid demonstration of data binding that users can easily refer to. Next, Additional information is required, which can include a simple, engaging tutorial, explanations of professional concepts, and intuitive attribute options with images. In addition, the interface should enable users to reflect on the data at any time. Last but not least, reconsideration of data binding options is needed. A simple solution is only allowing users to bind numeric data with numeric attributes and categoric data with categoric attributes.

From a long-term perspective, some future work, which may include extensions to NetPanorama grammar, can be done to improve the expressiveness and accessibility of Vistorian. First, allowing the creation of legends or other components can enhance the readability of diagrams. Then, advanced interaction techniques should be provided to enable deep customization rather than simple changes on individual elements. Other improvements can focus on further encouraging exploration. Currently, the prototype

supports a simple undo/redo feature in the toolbar, but more complex mechanisms can be applied. For example, Graphies [24] has an inspiring example of a timeline, which allows users to create multiple visualization branches. Users can continuously test different ideas without worrying about losing previous ones.

Bibliography

[1] JJ Allaire, Peter Ellis, Christopher Gandrud, Kevin Kuo, BW Lewis, Jonathan Owen, Kenton Russell, Jennifer Rogers, Charles Sese, C Yetman, et al. Package 'networkd3'. *D3 JavaScript network graphs from R*, 2017.

[2] BV Almende, Benoit Thieurmel, and Titouan Robert. Package 'visnetwork'. *Network Visualization Using 'vis. js' Library, Version*, 2(9), 2019.

[3] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Proceedings of the international AAAI conference on web and social media*, volume 3, pages 361–362, 2009.

[4] Alex Bigelow, Steven Drucker, Danyel Fisher, and Miriah Meyer. Reflections on how designers design with data. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, pages 17–24, 2014.

[5] Alex Bigelow, Steven Drucker, Danyel Fisher, and Miriah Meyer. Iterating between tools to create and edit visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):481–490, 2016.

[6] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics*, 15(6):1121–1128, 2009.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[8] Jean-Philippe Coene. sigmajs: An r htmlwidget interface to the sigma. js visualization library. *Journal of Open Source Software*, 3(28):814, 2018.

[9] Gabor Csardi, Tamas Nepusz, et al. The igraph software package for complex network research. *InterJournal, complex systems*, 1695(5):1–9, 2006.

[10] Max Franz, Christian T Lopes, Gerardo Huck, Yue Dong, Onur Sumer, and Gary D Bader. Cytoscape. js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311, 2016.

[11] Lars Grammel, Chris Bennett, Melanie Tory, and Margaret-Anne D Storey. A survey of visualization construction user interfaces. In *EuroVis (Short Papers)*, pages 019–023, 2013.

[12] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[13] Steve Hamersky. Tableau desktop. *Mathematics and Computer Education*, 50(2):148, 2016.

[14] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

[15] Monique WM Jaspers, Thiemo Steen, Cor Van Den Bos, and Maud Geenen. The think aloud method: a guide to user interface design. *International journal of medical informatics*, 73(11-12):781–795, 2004.

[16] Nam Wook Kim, Nathalie Henry Riche, Benjamin Bach, Guanpeng Xu, Matthew Brehmer, Ken Hinckley, Michel Pahud, Haijun Xia, Michael J McGuffin, and Hanspeter Pfister. Datatoon: Drawing dynamic network comics with pen+ touch interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[17] Nam Wook Kim, Eston Schweickart, Zhicheng Liu, Mira Dontcheva, Wilmot Li, Jovan Popovic, and Hanspeter Pfister. Data-driven guides: Supporting expressive design for information graphics. *IEEE transactions on visualization and computer graphics*, 23(1):491–500, 2016.

[18] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In

Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pages 1–13, 2018.

[19] Gonzalo Gabriel Méndez, Uta Hinrichs, and Miguel A Nacenta. Bottom-up vs. top-down: Trade-offs in efficiency, understanding, freedom and creativity with infovis tools. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 841–852, 2017.

[20] Giancarlo Perrone, Jose Unpingco, and Haw-minn Lu. Network visualizations with pyvis and visjs. *arXiv preprint arXiv:2006.04951*, 2020.

[21] Casey Reas and Ben Fry. Processing: programming for the media arts. *Ai & Society*, 20:526–538, 2006.

[22] Donghao Ren, Tobias Höllerer, and Xiaoru Yuan. ivisdesigner: Expressive interactive design of information visualizations. *IEEE transactions on visualization and computer graphics*, 20(12):2092–2101, 2014.

[23] Donghao Ren, Bongshin Lee, and Matthew Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE transactions on visualization and computer graphics*, 25(1):789–799, 2018.

[24] Hugo Romat, Caroline Appert, and Emmanuel Pietriga. Expressive authoring of node-link diagrams with graphies. *IEEE Transactions on Visualization and Computer Graphics*, 27(4):2329–2340, 2019.

[25] Arvind Satyanarayan and Jeffrey Heer. Lyra: An interactive visualization design environment. In *Computer graphics forum*, volume 33, pages 351–360. Wiley Online Library, 2014.

[26] Arvind Satyanarayan, Bongshin Lee, Donghao Ren, Jeffrey Heer, John Stasko, John Thompson, Matthew Brehmer, and Zhicheng Liu. Critical reflections on visualization authoring systems. *IEEE transactions on visualization and computer graphics*, 26(1):461–471, 2019.

[27] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.

[28] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2015.

[29] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE transactions on visualization and computer graphics*, 18(12):2431–2440, 2012.

[30] Beibei Si, Yuxuan Liang, Jin Zhao, Yu Zhang, Xiaofei Liao, Hai Jin, Haikun Liu, and Lin Gu. Ggraph: an efficient structure-aware approach for iterative graph processing. *IEEE Transactions on Big Data*, 8(5):1182–1194, 2020.

[31] Carson Sievert. *Interactive web-based data visualization with R, plotly, and shiny*. CRC Press, 2020.

[32] Andre Suslik Spritzer, Jeremy Boy, Pierre Dragicevic, Jean-Daniel Fekete, and Carla Maria Dal Sasso Freitas. Towards a smooth design process for static communicative node-link diagrams. In *Computer Graphics Forum*, volume 34, pages 461–470. Wiley Online Library, 2015.

[33] Fernanda B Viegas, Martin Wattenberg, Frank Van Ham, Jesse Kriss, and Matt McKeon. Manyeyes: a site for visualization at internet scale. *IEEE transactions on visualization and computer graphics*, 13(6):1121–1128, 2007.

[34] Yanyan Wang, Zhanning Bai, Zhifeng Lin, Xiaoqing Dong, Yingchaojie Feng, Jiacheng Pan, and Wei Chen. G6: A web-based library for graph visualization. *Visual Informatics*, 5(4):49–55, 2021.

Appendix A

Participants' information sheet

Project title:	Network Visualization Visual Editor
Principal investigator:	Benjamin Bach
Researcher collecting data:	Xiaoyang Chen
Funder (if applicable):	

This study was certified according to the Informatics Research Ethics Process, reference number 329009. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

The research team comprises three individuals: MSc student Xiaoyang Chen, who conducts the research primarily, his research partner, postdoc Xinhuan Shu, and his supervisor, Dr Benjamin Bach.

What is the purpose of the study?

The study will carry out a series of user tests on a network visualization visual editor interface we have designed, which allows users to design network visualizations (e.g., node-link diagrams) without programming. The aims are to evaluate the expressiveness and efficiency of the interface, and to collect users' feedback on it. This will help us identify the flaws that are likely to negatively affect the capability and user experience of the interface, and provide guidance on future improvement.

Why have I been asked to take part?

You have been asked to take part in the study because we hope people can use network visualizations as a tool for efficiency and creativity without programming expertise. Therefore, anyone with a minimum understanding of network visualizations is welcomed to take part in it.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time without giving a reason. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

You will take part in a user test about our interface. Before the test you will be asked some demographic questions and introduced to the editor. The test will be designed around completing certain/open-ended tasks on given datasets, during which we will record your interaction with the interface, time consumed and the outcome. You will also be asked to give some feedback after the test.

While set up to take approximately 30 minutes, the test may extend to 1 hour, depending on your wish to explore the interface and provide further feedback. The test will be conducted face-to-face at a pre-arranged venue.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

You will be able to get snacks and drinks for participating. You will also know more about network visualizations and tools to create it, which may be useful in the future.

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a maximum of 4 years. All potentially identifiable data will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name Your data will only be viewed by the researcher/research team Xiaoyang Chen, Xinhuan Shu and Benjamin Bach.

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will

be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Xiaoyang Chen at s2434689@ed.ac.uk. If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Xiaoyang Chen at s2434689@ed.ac.uk.

General information.

For general information about how we use your data, go to: edin.ac/privacy-research

Appendix B

Participants' consent form

Participants read and completed an consent form, as shown in *Figure B.1*, before participating in the user study.

Participant number:_____

Participant Consent Form

Project title:	Network Visualization Visual Editor
Principal investigator (PI):	Benjamin Bach
Researcher:	Xiaoyang Chen
PI contact details:	bbach@exseed.ed.ac.uk

By participating in the study you agree that:

- I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.
- My participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick yes or no for each of these statements.

1.	I agree to being audio recorded.	☐ **Yes**	☐ **No**
2.	I agree to being video recorded.	☐ **Yes**	☐ **No**
3.	I allow my data to be used in future ethically approved research.	☐ **Yes**	☐ **No**
4.	I agree to take part in this study.	☐ **Yes**	☐ **No**

Name of person giving consent	Date dd/mm/yy	Signature

Name of person taking consent	Date dd/mm/yy	Signature

THE UNIVERSITY of EDINBURGH informatics

Figure B.1: The Participant Consent Form