The Path to Web-based Visual Network Exploration

Dávid Debnár, Sasha Gielis, Theo de Leeuw, Julian Dziegielewski and Jeroen Oerlemans

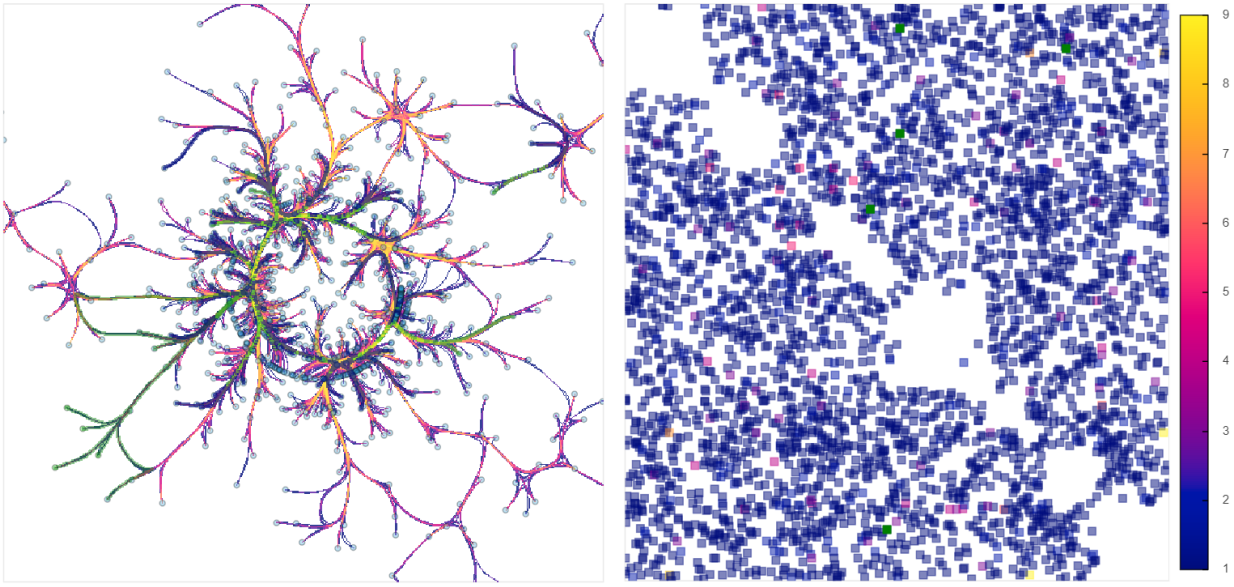

Fig. 1: Both supported views, the node-link diagram using a circular layout on the left and adjacency matrix on the right.

Abstract— Data exploration is best facilitated by the synthesis of algorithm-based representations and human-computer interaction, multiplied by the mobility of information gained through online collaboration and sharing of results. Given many emerging real-world systems reduce to networks that impact daily life and decision-making, an open and accessible approach to information visualization is required. The research hence describes the techniques used in developing a web-based, interactive data exploration and visualization tool for directed network graphs.
Drawing on research into human-computer interaction for information visualization, and algorithms for representing network graphs, specific choices of the tool design are discussed. The supported data format, for upload of datasets to be analyzed, is an adjacency matrix of weighted edges, based on the most wide-spread conventions. Networks are visualized using two metaphors, the node-link diagram, and the adjacency matrix. Interactions, as categorized by user intent, include exploration by pan and zoom, filtering, re-configuring of elements in views, selection and consequent highlighting of connected elements. Users can reconfigure the vertices of the node-link view into a circle. This is achieved using the Bokeh library for Python deployed as an embedding into a Flask website application.
The aim of this paper is describing techniques used in deploying a tool which can further expand the software landscape of online interactive data exploration.

Index Terms—Information visualization, Directed networks, Interaction, Collaboration, Visual exploration.

1 Introduction

Expanding the ability to visualize network data is an ever-increasing need in the era of being surrounded by various networks. Every system with a flow of information can be described by a network, from brain-connectivity related to Alzheimer's disease [24], to semantic networks in linguistics or social networks of societies. Hence a versatile tool can have far-reaching applications. With a personal computer in every pocket, one might be inclined to explore data of the physical or digital networks they are part of even outside academia, for their own daily decision-making.

Big-data collection has only been rising in recent years, and though paralleled by the increase in applications of machine learning or other statistical-learning methods, the human brain, when aided by interactive tools, still leads in terms of pattern-recognition [17]. Given the difficulty of anticipating possible uses for in-flowing data, the decision to gather is often made before intent of analysis, and so network data, among others, is being over-collected and under-analyzed.

Visual representations of network data coupled with interaction techniques, allow for a deeper understanding of behavior within networks, but often require a steep learning curve, are limited to simple networks or single-purpose [25]. Scaling to networks with a high vertex-count or with complex, dense connectivity is a key requirement when considering applications in areas like brain-imaging and social-networking. The analysis of networks is therefore bounded by the entry-requirements and the subsequent iteration-speed of individuals.

The paper will describe the techniques used in developing a web-based, interactive information-visualization (InfoVis) tool for the exploration of directed graph networks. It aims to describe InfoVis approaches harnessed by the capability to share results via web-based tools, which open the field of data analysis to a wider range of actors

and audiences thus giving rise to more opportunities for the synthesis between the data gathered and human brains available as pattern-recognizing machines.

2 Related Work

Visualization as a graph, network, or adjacency matrix is an integral part of data and computer science [11]. It is used in applications and domains [18].

Just like people are interacting with each other creating networks, computer systems have the same relation between co-existing functions in the program. There are countless more examples of our world. All of which have a common structure of relations between themselves. They can be distinguished by their form. When the relation between to elements has a direction(either one or both way) we call them directed and if a relation has no indicators, it is undirected. In graph theory, we have edges as the relations and vertices which are our elements. Such a set of vertices and edges is called a graph. We declare a graph by G=(V, E). There are many types of graphs. Already mentioned directed and undirected (the name comes from a type of relation between the elements). If the edges contain the weight of the relation between vertices we call such graphs weighted. The weight can be for example; a number of orders a customer placed in a shop. As opposed to graph terminology when we focus on a visual representation of the data we refer to graph and as to visualization and it is components to nodes(vertices) and links(edges). This type of data visualization is called a node-link diagram, where a node is displayed by a circle and a link by a line.

It has been invented by Leonard Euler in 1736 when he was solving 'The Seven Bridges of Königsberg' problem [21]. Euler's discovery is considered to be the first theorem of graph theory. Even though during Euler's time it was the best visualization for this data, these days the node-link diagrams are not efficient due to constantly enlarging datasets which cause problems such as visual clusters of links crossing each other, node overlaps and link distances [20] [14]. Although they can be reduced by applying different layouts; radial, hierarchical or force-directed, even then there are visual 'hairball' effects filling entire inter-node space [15].

Those problems can be resolved either by interactions or by switching to/with the help of another visualization method that resolves visual problems of the node-link diagram. One of them is an adjacency matrix [16]. It is a square matrix in which elements indicate whether pairs of vertices are adjacent or not. This type of graph eliminates line crossings and overlapping nodes. However, while using the adjacency matrix at first ordering of the data has to be done, otherwise, it will not be possible to identify groups or highly connected elements, showing patterns of higher order [10].

The web-based network visualization and dataset management tool presented in this report enables users to explore either datasets from multiple different formats and techniques mentioned before. Networks being visualized by the node-link diagram and the adjacency matrix can be reordered using several algorithms and customized in their rendering style, colors, interactions, and themes. All this is further elaborated in the next Sections.

3 Data

As mentioned before, the type of data that will be visualized with this tool is network data. Within this data type, there can be some variations. Directed networks and weighted networks are the network types one might encounter.

A directed network has relations that may only hold in one direction. For example, Alice might have sent a message to Bob, but Bob might not have sent a message to Alice. Alice is then related to Bob, but Bob is not related to Alice.

A weighted graph is a graph where the relations have weights. A relation between one pair of nodes might have a higher weight than another. To give an example, Alice might have sent ten messages to Bob and only one message to Charlie. The relation between Alice and Bob has a higher weight than the relation between Alice and Charlie.

3.1 Data Format

There are several formats that one can use to represent all of these networks. In this section two of them will be discussed, and we will see which of the two formats was chosen to store the data to be visualized in.

The first format that will be discussed in this section is a compact one. It involves simply writing down the relations in a list:

$$\begin{array}{lll} A,B & & A,B,C \\ A,C & \text{or even more compact:} & B,A \\ B,A & & \vdots \\ \vdots & & \end{array}$$

The first node is the source of the relation, the other node(s) is the target of the relation. Weights can be implemented, for example, by repeating the same relation multiple times. The weight could also be added after each relation. Storing networks this way is very space efficient, however, it is not the easiest to work with. This is because one has to extrapolate how many nodes are included in the network. It also only includes nodes that have a relation to other nodes, making the node-link diagram seem more populated than it might actually be. For these reasons, it was chosen not to go with this format.

Another way of storing networks is by using a so-called adjacency matrix:

$\downarrow$ nodes$\rightarrow$	A	B	C
A	0	1	1
B	1	0	0
C	0	0	0

Each source and target combination is displayed, however, only the cells where the value is nonzero are included in the network. To implement weights, simply change the cell from a 1 to its weight. This format is less space efficient than the other one, however, it is easier to work with. This is because it is known beforehand how many nodes to include in the network. It has thus been decided to use this format to store the data in. Coincidentally, the datasets that were provided were formatted this way already.

3.2 Data Upload

To be able to create several visualizations from a dataset, the uploaded file needs to be written in one of two specific formats. As mentioned before, it was chosen to store the datasets in the adjacency matrix format. However, edge list datasets are supported as well. The dataset can be uploaded in various file types. The supported file types are CSV, TXT, XLS, XLSX, XLSM, and JSON. To see how to adhere to the adjacency matrix or edge list format in their favourite file type, users can look at the documentation page. The upload functionality is achieved through a library called 'Flask' [2], in conjunction with the 'os' module [4]. The former of these is a web framework written in Python. It allows for full website functionality management. The latter is used for file management and interaction on the server. The upload page interface is written in HTML. The code calls on a specific function in the Flask server, which then saves the file to a temporary directory for pre-processing. Since the provided datasets already have the adjacency matrix format, those files are not pre-processed much before creating the visualizations.

3.3 Data Pre-processing

If the uploaded dataset is accepted, pre-processing can commence. The dataset is read into a pandas DataFrame [5], so that it can be visualized more easily. Depending on the file type that was uploaded, making the DataFrame happens differently. For CSV and TXT files, the method 'read_csv' will read these file types into a DataFrame. It can also read zipped CSV and TXT files, if the dataset is the only file in the compressed folder. This function also has an argument for a custom separator. Users can enter a separator before pressing the

'Upload' button, and it will be used here. For the Excel Sheet formats, the function 'read_excel' is called. This function should also support decompression, but it was chosen not to support this, due to issues with getting it to work. Finally, to read JSON files, the 'JSON' module in Python was utilized. The file is loaded into a Python dictionary using 'json.load', and then into a pandas DataFrame, using the pandas 'DataFrame.from_dict' method. While pandas has a 'read_json' function, there were some issues with reading nested JSON files, which are needed for the adjacency matrix.

If the user indicates that they are uploading an edge list dataset, the dataset will now be converted to the adjacency matrix format. This is done using a custom algorithm.

```
def edli2adm(df):
    nodes = df.start.unique()
    np.append(nodes, df.end.unique())
    nodes = list(dict.fromkeys(nodes))
    adm = pd.DataFrame(index=nodes,
        columns=nodes)
    for i in range(len(df['start'])):
        adm[df['start'][i]][df['end'][i]]
            = df['weight'][i]
    adm = adm.fillna(0)
    return adm
```

Once a dataset is read into a DataFrame, it is saved to the server using the 'cloudpickle' [1] module. This module allows for the saving and loading of Python objects to the server with a .pkl extension. Through the use of this module, it was not necessary to process the datasets into a DataFrame every time the visualization was launched, but the DataFrame could be loaded directly. As the DataFrame is saved to the upload folder, the original dataset is removed from the temporary folder.

3.4 Dataset Management

Once the upload process is finished, the filename will be displayed in the 'Uploaded Files' section. This is made possible by the 'listdir' method of the os module. From this list the .pkl extension was removed from each filename, so as to not create confusion for the user. Through Flask, the list of filenames is passed to the upload page, which then displays them. Each filename is a hyperlink to the page with the corresponding dataset as the visualization. Next to each filename is a download button. When users click on this button, they will get a CSV file with the dataset in an adjacency matrix format. This is because of the way uploaded datasets are stored on the server. The download functionality is achieved by loading the DataFrame object from the server, and then converting it to a CSV file. This is done using the 'DataFrame.to_csv' method of pandas. The Flask method 'send_file' serves this CSV as an attachment to the user. Users can also delete any dataset from the server, made possible by the 'remove' method from the os module.

4 Visualization Tool

The library that is used by the tool to create the visualizations is called 'Bokeh' [12]. Bokeh is a library for Python and has a wide set of options for visualization. It not only allows for graph drawing and matrix creation but also allows for built-in interactions. The interactions that are implemented into this visualization tool will mostly consist of Bokeh's built-in interaction features.

4.1 Bokeh's Interface

Bokeh uses a special interface. The node-link diagram in figure 2 is the representation in Bokeh. On the right of the node-link diagram, we see an interface consisting of three icons, two of which are interactive tools provided by Bokeh. The top icon is the Bokeh logo itself, redirecting to Bokeh's website. In this example, a hover tool and a wheel zoom tool have been added to the interface.

Let us consider the hover tool again to explain the interactive tools. As can be seen in figure 3, hovering over an edge or node will display what the developer of the tool wants to be displayed. We can add these

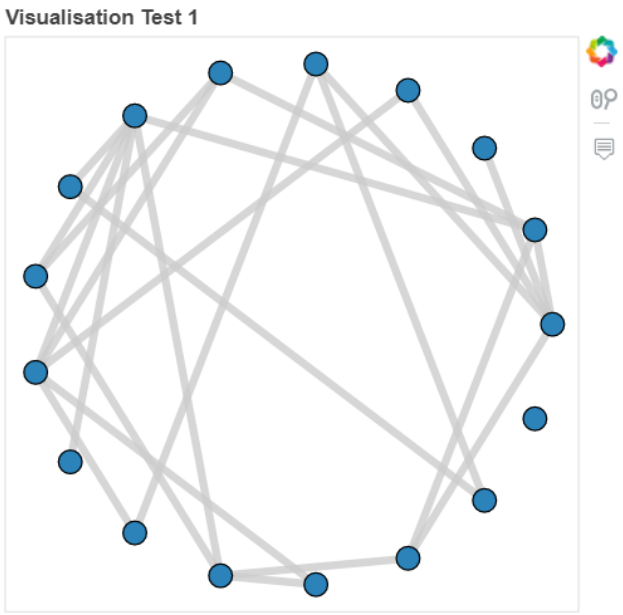

Fig. 2: Node-link diagram of a specific dataset.

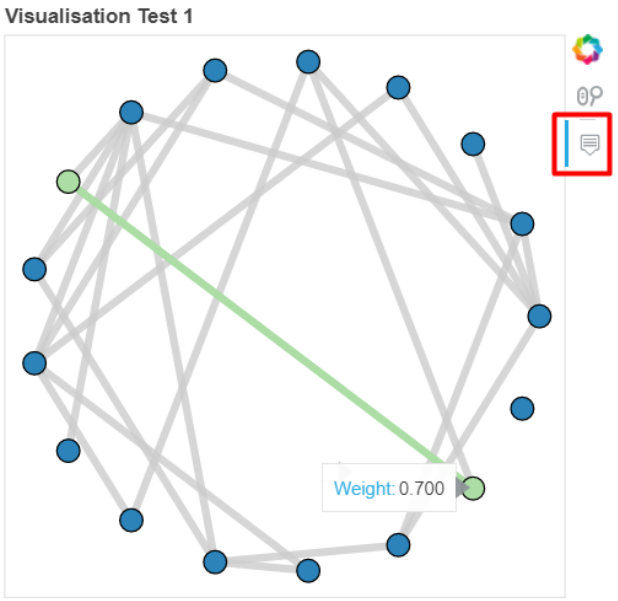

Fig. 3: Node-link diagram when using the hover tool.

tools easily. Suppose a variable G is set to be a HoloViews graph [23], then this is how we set the hover tool:

```
import pandas as pd
import numpy as np
import holoviews as hv
from holoviews import opts
hv.extension('bokeh', 'matplotlib')

h = hv.Graph()
opts.Graph(tools=['hover'])
```

Now there is a Bokeh interaction implemented for our visualization, but there are many more tools that can be used. Our tool uses the hover tool, as shown above in Figure 3, which allows you to hover over edges, nodes, and the adjacency matrix. Another implemented

interaction is the box select tool, which allows for selecting an area in the visualization resulting in selecting the corresponding edges and nodes in the node-link diagram or fields in the adjacency matrix.

4.2 The Tool's Interface

As for the tool's interface, a toolbar is present to customize the Bokeh visualization to their liking

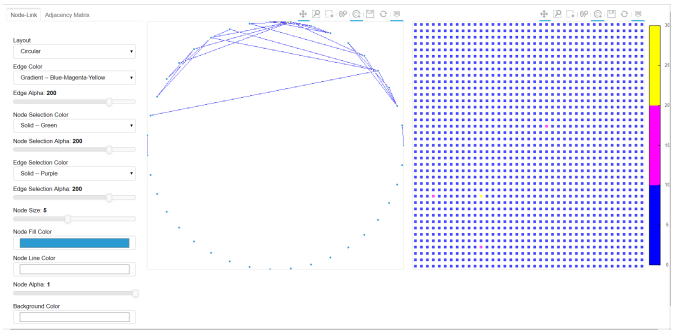

Fig. 4: The tool's interface for the visualizations.

In Figure 4 we see two visualizations: a node-link diagram and an adjacency matrix.

4.2.1 Node-Link Diagram

For the Node-Link Diagram, an edge list is calculated. A user has the option between different orderings: Circular

4.2.2 Adjacency Matrix

For the adjacency matrix a list of edges and names are calculated. The user has to specify an ordering method using the interactions described in the Interactions section. To visualize these orderings a diagonal is computed. To illustrate this effect take a chess board. If we look at its diagonal and want to swap row and column 4 (Yellow) with 5 (Blue) (The purple edge is to illustrate changes):

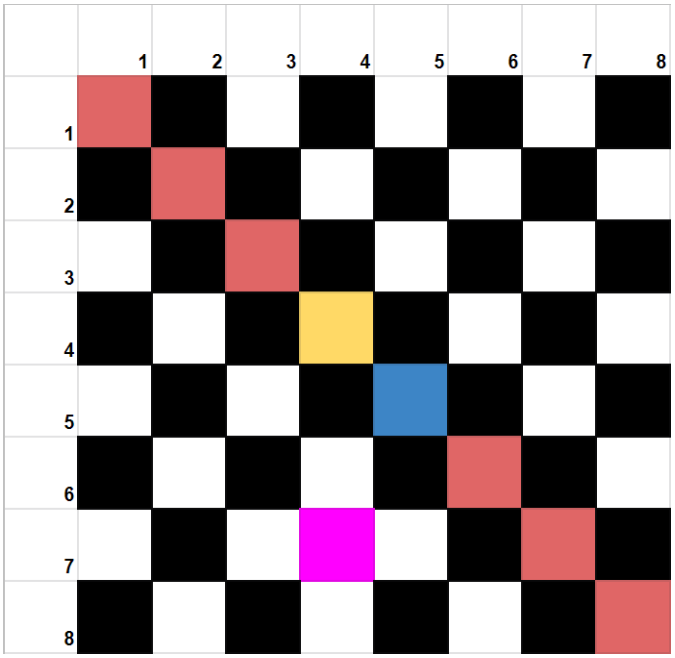

Fig. 5: Chess representation of a diagonal.

Figure 5 can be restructured so the diagonal forces the matrix to be computed differently by swapping index 4 and 5, resulting in:

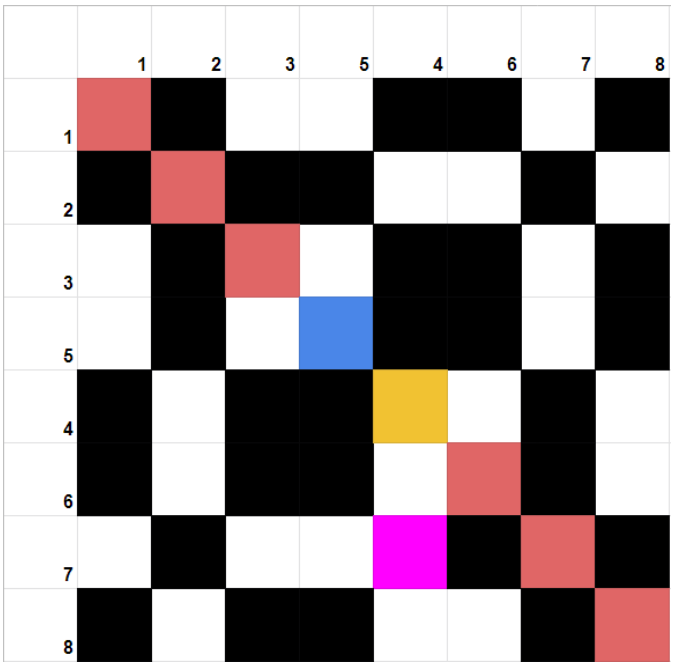

Fig. 6: Chess representation after diagonal sorting

As seen in Figure 6 the matrix is successfully reordered by only looking at the diagonal. Applying this logic to our algorithms we can sort efficiently and code our orderings by returning a diagonal. The orderings are Sorted, Hierarchical Clustering, Reverse Cuthill-Mckee and Fiedler Vector Clustering. The working of these reordering strategies will be explained in Section 4.3 (Algorithms) and examples of them can be seen in Figure 7.

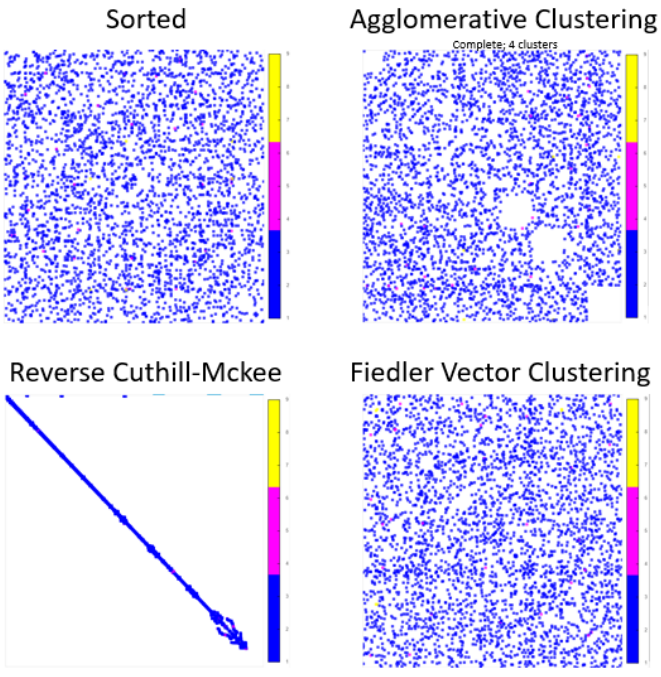

Fig. 7: The differerent Adjacency Matrix orderings

After the correct ordering is applied we draw the ordering and then the matrix with Holoviews [23]. Resulting in an adjacency matrix depicted in Figure 4 using the Sorted ordering.

The toolbar on the left in Figure 4 will be representing the interface providing various interactions.

4.3 Algorithms

Initially, the nodes in the adjacency matrix are placed in alphabetical ('Sorted') order. There are various ways to reorder the nodes [10], of which three algorithms are implemented within this tool: Hierarchical Clustering, Reverse Cuthill-McKee and Fiedler Vector Clustering. With these reordering strategies, users can change the order of the nodes in the adjacency matrix to be able to discover possible patterns in the data.

4.3.1 Hierarchical Clustering

The Hierarchical Clustering algorithm [] builds a hierarchy of clusters, which can be represented by a dendrogram []. In our tool the hierarchy is created in an agglomerative way, which means that initially each node is placed in its own cluster and while moving up the hierarchy, pairs of clusters are merged. The way the clusters are combined depends on the linkage type that is used to determine the distance between clusters.

This tool enables the user to choose one out of three linkage types. Complete-linkage clustering merges the two clusters containing the nodes (one in each cluster) with the largest distance between each other. Another linkage type is single-linkage, which combines the two clusters containing the nodes (one in each cluster) that are closest to each other. The last option is average linkage, which is a balance between the two extremes complete- and single-linkage. For both complete- and single-linkage there are optimally efficient algorithms in $O(n^2)$ time, known as 'CLINK' [13] and 'SLINK' [22], respectively. Without making any assumptions on the input, the best possible runtime for average linkage is $O(n^2 log(n))$.

Next to the linkage type, users can also select the number of clusters the algorithm should create. A dataframe of the dataset together with the two inputs are used for the *AgglomerativeClustering* function provided by the library 'scikit-learn' [6]. Another parameter for this function is 'affinity', which determines the calculation of the distances. Since the tool assumes the datasets to have an adjacency matrix format, the values in the dataframe are already equal to the distances between the corresponding nodes. This is called the distance matrix of the graph, which can be handled by the function if the 'affinity' parameter is set to 'precomputed'.

After creating the clusters, the algorithm places the clusters next to each other and returns the new order of the nodes. As creating clusters already suggests, this algorithm tends to produce grouping patterns.

4.3.2 Reverse Cuthill-McKee

The Cuthill-McKee algorithm [] is a variant of the Breadth-First Search (BFS) algorithm. It starts with the node with the lowest degree and then visits the neighboring nodes in order from lowest to highest degree. It aims to minimize the bandwidth of the adjacency matrix. The Reverse Cuthill-McKee algorithm consists of the exact same operations, but in the end the order of the nodes is reversed, which does not have an influence on the bandwidth of the matrix. This small modification often yields an ordering that is more efficient than the original algorithm [19].

The Reverse Cuthill-McKee algorithm of the tool has a dataframe as input and uses the function *reverse_cuthill_mckee* from the 'SciPy' library [7] to reorder the nodes. To be able to use this function, the dataframe first needs to be converted to either a Compressed Sparse Column (CSC) [8] or a Compressed Sparse Row (CSR) matrix [9].

4.3.3 Fiedler Vector Clustering

The *fiedler_vector* function provided by the library 'NetworkX' [3] only supports connected graphs as input. Since the datasets are not necessarily connected, this function should not be used by the tool. So in contrast to the previous two algorithms, the application does not use a library to directly calculate the result for the Fiedler Vector Clustering. The pseudocode can be found in Algorithm 1 with a dataframe of the dataset as input and the new order of the nodes as output.

This algorithm uses linear algebra techniques to create the Fiedler vector of a dataframe []. It first transforms the dataframe into a Laplacian matrix $L = D - A$, where D is the degree matrix and A the adjacency matrix (dataframe). Then the eigenvalues are computed and the eigenvectors belonging to them. Now the Fiedler vector is the eigenvector corresponding to the second lowest eigenvalue. The values of this Fiedler vector can be used to partition the nodes into two clusters: the nodes corresponding to the negative values and the nodes corresponding to the positive values. But to create more of a pattern in the adjacency matrix, the algorithm orders the nodes in the order of the corresponding sorted values in the Fiedler vector.

Calculating the Laplacian matrix involves subtracting the *nxn* degree matrix from the *nxn* adjacency matrix. This takes $O(n^2)$ time, which makes it the most time-consuming operation of the algorithm. Therefore, the asymptotic runtime of Algorithm 1 is $O(n^2)$.

Algorithm 1 Fiedler Vector Clustering

```
dataframe ← dataframe of the dataset given as input
laplacian_matrix ← dataframe transformed into a Laplacian matrix
eigenvalues ← set containing all eigenvalues of the Laplacian matrix
eigenvectors ← set containing all eigenvectors of the Laplacian matrix
seen, unique_eigenpairs ← empty sets
for (eigval, eigvec) in (eigenvalues, eigenvectors) do
    if eigval in seen then
        continue with next loop iteration
    end if
    add eigval to seen
    add the pair (eigval, eigvec) to unique_eigenpairs
end for
sort unique_eigenpairs on eigenvalues
fiedler_vector ← second eigenvector in unique_eigenpairs
fiedler_labels ← empty dictionary
fill fiedler_labels with the labels of the nodes and assign to them their corresponding value in fiedler_vector
sort fiedler_labels on values
new_order_of_nodes ← order of the nodes in fiedler_labels
return new_order_of_nodes
```

5 Interactions

Among the main goals of this tool is facilitating exploratory data analysis. While ordering algorithms provide a necessary basis for data exploration in the form of representations, they are only part of a continuous back-and-forth process between a human user and their data analysis tools, compelling the need for further interactions in a data exploration tool. Interactions considered in this tool can be categorized based on the model of notion of user intent, which include *filter*, *explore*, *select*, *reconfigure*, and *connect* [26].

As previously discussed, the graphical user interface of the visualization consists of a tabbed interaction panel docked to the left and two views side-by-side, each with their own visualization. The view on the left is a node-link diagram, which encodes relations of vertices and edges, and adds a spatial channel in the form of position of nodes, and by extension position of links. The view on the right is an adjacency matrix, which encodes edges as squares on a grid, and vertices as row and column labels. Both views share source data, which is reflected in several interactions, but the rest operate independently.

Filtering tools modify the source data loaded in memory, and as such are reflected on both views simultaneously. These include taking a random subset of vertices and hence edges prior to visualizing, for exploring large datasets dynamically, if limited by performance when viewing the entire network.

General exploratory interactions are pan and zoom, essential for discovering patterns at different layers of emergence, as well as making sense of large or complex networks in detail. These are provided as part of tools instantiated in Bokeh plots out-of-the-box, specifically panning a view by clicking and dragging over a view using a pointer, activated by default, and zoom using a rectangular selection. Further,

the mouse scroll-wheel zoom Bokeh tool is instantiated, providing a more sensitive zoom alternative.

For selection, box select, and hover is available as part of Bokeh's toolset. Box select allows a user a rectangular selection of the nodes channel in the node-link diagram and edge channel in the adjacency matrix. The selected items are highlighted by a different colour and remain highlighted until another selection is made. The hover tool, activated by default, displays information about the underlying item in a floating bubble, e.g. vertex labels or edge weights. For each selected node in the node-link diagram, the connected links are highlighted as well, hence selection in this network visualization is inherently linked with finding connections.

6 Application Example

The goal of this visualization tool is to aid users in discovering more about their data. Network data can be messy to analyze, due to the many possible relations. This tool aims to make it simpler to do so.

6.1 Landing Page

Upon connecting to the website, users will land on a welcome page. On this page, users can look at the group members and a video about the tool. Users also get the option to look at this very paper. In the top right is a menu bar with a link to the landing page, dataset and visualization page, and a documentation page.

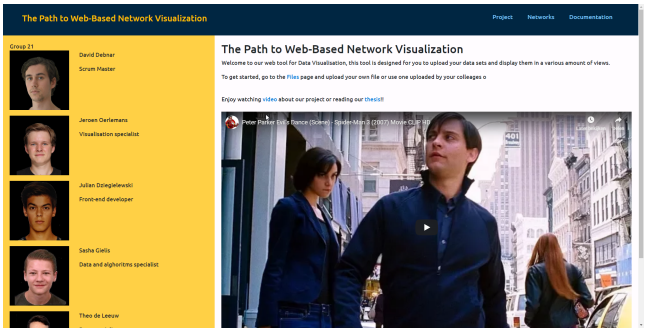

Fig. 8: A screengrab of the landing page. The group is depicted on the left, and the video is shown on the right.

Upon pressing one of these buttons, users will be redirected to the appropriate page.

6.2 Documentation

On the documentation page users can find information about the website and how it operates. At first, users will land on a page with general information about the tool. On the left hand side are links to pages of the documentation that deal with more in-depth topics. For example, there is a page documenting how to adhere to the adjacency matrix or edge list format for each supported file type. Users can also look at the documentation of the visualization to see what each button does.

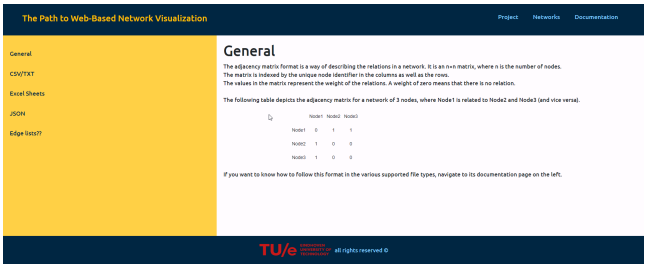

Fig. 9: The documentation page. In this picture, the general documentation is shown.

6.3 Networks

The 'Networks page' is the page every user will come across. It is where the user can find datasets uploaded by other data scientists, and also upload their own. On the right hand side, there is some compact information about the upload tool and what it supports. On the left is the upload tool and the list of uploaded files. The upload tool has some options for a custom separator and the format of the dataset to be uploaded. Below it, the filenames of the uploaded datasets are listed. Next to these names are two buttons. The user can press either of these buttons to respectively download or delete the dataset listed on the left. When users click on the filename of a dataset, the visualization is launched.

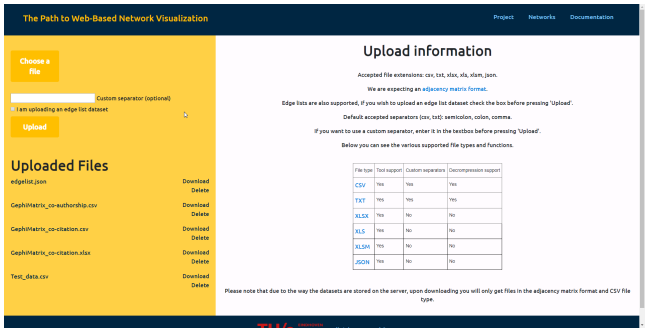

Fig. 10: The networks page. Here, users can upload a dataset or select one from the previously uploaded datasets.

6.4 Visualization

There are two visualizations side by side. On the right is the adjacency matrix visualization. By default, its axis labels are hidden, but users have the option to show them on any side they want. Next to the adjacency matrix is the colour bar, which explains the colouring by weight of relations in the adjacency matrix. The colour scheme can be changed by the user. On the left of the adjacency matrix is the node-link diagram visualization. It shows all the relations as lines (links) between two circles (nodes). The lines are coloured by weight as well.

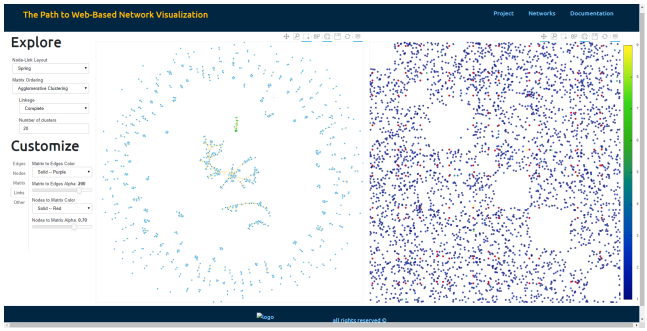

Fig. 11: The visualizations page. On this page, the dataset is visualized in two ways. The visualizations can be manipulated to the users' liking.

Above each visualization is a toolbar with interaction options. The default enabled interactions are pan and hover. Panning allows the user to move the graph around, and hovering over a node or link will show information about it. Other interaction options are zoom and select. Zooming lets the user get up-close to their data. Users can select a group of nodes by using the box select tool, or they can tap on a node or link to select it. Once a selection has been made, the relevant relations in the adjacency matrix are highlighted. If the user has found something interesting, they can save a picture of each visualization to their hard drive.

On the left of the visualizations, there is a dashboard with options for orderings, colouring, and renderers. The user can choose between

four layouts for the node-link diagram, and four clustering methods for the adjacency matrix. Some clustering methods have paramaters that the user can change. The colour and size of edges and nodes can be changed to the users' liking.

7 Discussion and Limitations

Even though the tool is fully operational, there are things that can be improved. These limitations will be discussed in the following Subsections; Performance, Visualizations, File Upload and Graphical User Interface (GUI).

7.1 Performance

The performance of the tool is dependent on the size of the file that the user chooses to visualize. It needs some time to be processed before the visualization will show up. For very large datasets the waiting time can take up to a minute. This is mainly due to the different asymptotic time complexities of the various layouts and ordering algorithms for the visualizations.

Since this is meant to be an interactive visualization tool, users should be able to obtain insight into a specific dataset using certain interactions. These interactions make the tool powerful but are also likely to influence its performance. This is another point of improvement because, after the visualizations of a dataset are displayed on the screen, the tool responds quite slowly to the interactions. This problem is also related to the size of the selected dataset. To make the tool truly interactive, it should nearly instantly respond to commands by the user.

7.2 File Upload

Currently the tool does not support networks which have data in multiple files.

7.3 Visualizations

The tool supports a finite amount of layouts for the visualizations. Possible layouts for the node-link diagram are radial/circular, Fruchterman-Reingold, Kamada-Kawai and Spring. The adjacency matrix can be reordered with the algorithms Hierarchical Clustering, Reverse Cuthill-Mckee, Fiedler Vector Clustering and Sorted.

7.4 Graphical User Interface

The GUI of the tool is basic. This is probably the least important limitation of the application since it does not necessarily make the tool poor to use. Instead, the navigation between the different parts of the tool is clear and easy to use. Just like Leonardo da Vinci said: 'Simplicity is the ultimate sophistication'.

8 Conclusion and Further Work

This paper's purpose is to describe our web-based visualization tool, as well as everything behind it, in great detail. It explains how a dataset is accepted and transformed into an interactive visualization with two different overviews: a node-link diagram and an adjacency matrix. A lot of attention has been paid to the structure, implementation, and functioning of the application, and many of its purposes have been mentioned. Despite reaching a lot of the goals that were originally set, the tool is far from perfect and there is a lot of room for improvement.

For starters, only datasets of a specific file type and format are accepted hence adding more flexibility to the data upload would be beneficial. Visualizing with custom colors is another feature that is not supported. Different colors would make setting apart the data easier in addition to helping people with color vision deficiency. Lastly, the user may actually need to do a lot of work in order to obtain a visualization that provides information. Hence automatic filtering and suggestions that could aid the user would save time while making the tool easier to work with. While there are great ideas for further development it should be noted that what has been mentioned barely scratches the surface of what is possible.

Acknowledgments

References

[1] cloudpickle - extended pickling support for python. `https://www.pydoc.io/pypi/cloudpickle-0.3.1/autoapi/cloudpickle/index.html`. Date accessed: June 2019.

[2] Flask (a python microframework). `http://flask.pocoo.org/`. Date accessed: May 2019.

[3] Networkx (a python library). `https://networkx.github.io/documentation/stable/`. Date accessed: Jun 2019.

[4] os - miscellaneous operating system interfaces. `https://docs.python.org/3/library/os.html`. Date accessed: May 2019.

[5] pandas - python data analysis library. `https://pandas.pydata.org/`. Date accessed: May 2019.

[6] Scikit-learn (a python library). `https://scikit-learn.org/stable/`. Date accessed: Jun 2019.

[7] Scipy (a python library). `https://www.scipy.org/`. Date accessed: Jun 2019.

[8] Scipy csc matrix. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csc_matrix.html`. Date accessed: Jun 2019.

[9] Scipy csr matrix. `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.csr_matrix.html`. Date accessed: Jun 2019.

[10] M. Behrisch, B. Bach, N. Henry Riche, T. Schreck, and J.-D. Fekete. Matrix reordering methods for table and network visualization. *Comput. Graph. Forum*, 35(3):693–716, June 2016. doi: 10.1111/cgf.12935

[11] M. Burch and D. Weiskopf. An interactive visualization tool for dynamic graphs, node ...

[12] B. contributors. `https://bokeh.pydata.org/en/latest/`, Apr 2013.

[13] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, Jan 1977. doi: 10.1093/comjnl/20.4.364

[14] C. Dunne and B. Shneiderman. Improving graph drawing readability by incorporating readability metrics: A software tool for network analysts. Technical report, 2009.

[15] D. Edge, J. Larson, M. Mobius, and C. White. Trimming the hairball: Edge cutting strategies for making dense graphs usable. *2018 IEEE International Conference on Big Data (Big Data)*, pp. 3951–3958, 2018.

[16] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. *IEEE Symposium on Information Visualization*, pp. 17–24, 2004.

[17] D. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):18, Aug 2002. doi: 10.1109/2945.981847

[18] T. v. Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. v. Wijk, J. Fekete, and D. Fellner. Visual analysis of large graphs: Stateoftheart and future research challenges, Apr 2011.

[19] W.-H. Liu and A. Sherman. Comparative analysis of the cuthill–mckee and the reverse cuthill–mckee ordering algorithms for sparse matrices, Apr 1976.

[20] R. Rosenholtz, Y. Li, Z. Jin, and J. Mansfield. Feature congestion: A measure of visual clutter, Jun 2006.

[21] T. Rz. Mathematical explanations in euler's konigsberg, 2014.

[22] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, Jan 1973. doi: 10.1093/comjnl/16.1.30

[23] J.-L. Stevens, P. Rudiger, and J. A. Bednar. Graph.

[24] F. Vecchio, F. Miraglia, F. Piludu, G. Granata, R. Romanello, M. Caulo, V. Onofrj, P. Bramanti, C. Colosimo, P. M. Rossini, and et al. "small world" architecture in brain connectivity and hippocampal volume in alzheimer's disease: a study via graph theory from eeg data, Mar 2016.

[25] M. Xia, J. Wang, and Y. He. Brainnet viewer: A network visualization tool for human brain connectomics. *PLoS ONE*, 8(7), 2013. doi: 10.1371/journal.pone.0068910

[26] J. S. Yi, Y. A. Kang, and J. Stasko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):12241231, 2007. doi: 10.1109/tvcg.2007.70515