Graphion: A Customizable Graph Visualization Tool

Anton Vellikok, Sam Baggen, Steven van den Broek, Tim van de Klundert, Tom Udding, Yuqin Cui and Yuqing Zeng

G R A P H I O N

YOUR DATA VISUALIZATION EXPERT

Fig. 1. The banner of our team and central part of the homepage of the website

Index Terms—Visualization, Node-link diagram, Adjacency matrix, Interactions

1 Abstract

This paper describes our online data visualization tool. It allows anyone with access to the internet to upload data sets and visualize them in their browser using various kinds of graphs. The tool is written in Python and makes use of various packages like Bokeh, HoloViews and NetworkX. These packages enables the user to view the node-link diagram and/or adjacency matrix of the uploaded data set. These views also allow several interactions with the data set to study it more in depth, like zooming and panning. Furthermore, our tool features a filtering step to let the user decide what part of the data is used to generate the visualizations and thus what the focus is of the visualizations.

2 Introduction

Humans are very bad at reading and interpreting large data sets, they are completely lost as soon as they see a page that is mostly filled with numbers and text. But when those numbers are turned into colourful graphs, everything is suddenly much clearer and easier to read, according to a study conducted by Vogel et. al. [8]. That is because the human eye is simply not that good at reading text and numbers. Luckily, computers are much better suited for this task as they can easily process hundreds of pages within a second.

Data visualizations combine the power of human intelligence and computer data processing power to analyze large amounts of data very well, which make them very powerful. Since large amounts of data are readily available nowadays, visualization tools are required to find any insights into the data forest. Our application is such a tool. It can both show an overview of the entire data set and zoom in on specific subsets of the data.

According to Okoe et. al. [7] the two most popular ways to visualize a graph are via a node-link diagram and an adjacency matrix. The reason to both implement a node-link diagram and an adjacency matrix is because node-link diagrams are very useful for path-related tasks, as they outperform an adjacency matrix to solve path-related tasks except for very large graphs as is shown in a study by Ghonheim et. al. [5]. In that same study, Ghonheim et. al. also show that adjacency matrices are more useful for finding specific nodes and specific links compared to node-link diagrams. However, more visualizations exist like a 3D node-link diagram.

3 Related Work

Our tool builds upon graph theory, as described in various books and articles by Matoušek and Nešetřil [6], Bollobás [2], Cormen et. al [3], Diestel [4] and many more. Furthermore, to enable the user to gain insights into the data that is visualized, we have implemented a few interactions according to the categories of interaction defined in the paper by Yi. et. al. [9].

4 Technical Specifications

In order to build the tool, we have chosen to use Python and the Pyviz environment. In particular, the packages Bokeh, Panel, HoloViews, NetworkX, Plotly, Pandas, Flask and Nginx were predominantly used. Bokeh, NetworkX, Plotly and HoloViews provide the visualizations. Panel is the library that enables us to have a multi-view system and makes the matrix re-orderings possible. Pandas provides an easy way to manipulate and store data while Flask and Nginx provide support for running Python code in an online environment. We chose these packages because Bokeh (which is part of HoloViews) seemed better than other packages for its simplicity for rendering graphs and providing native support for several types of interactions, like panning and zooming. Furthermore, HoloViews has native support for NetworkX which makes it easier to read data and convert it into a directed graph, something that was more complicated whilst only using Bokeh. Additionally, we used Pandas for data processing as all team members have worked with this package before and it is an easy way to store and manipulate tabular data. As the data that is accepted by our tool is in the adjacency matrix format, which is tabular, it matches perfectly with Pandas. Our back-end, which is all python code, uses Bokeh, NetworkX and Plotly to serve the graphs based on the data stored in Pandas. Then HoloViews combines Bokeh and NetworkX and instructs Bokeh to convert the python code into JavaScript which in turn is executed by the web browser and gets pushed to the front-end.

The front-end (the website) itself is mostly written in HTML and CSS. The website has a homepage, where the user can get to know the people behind the tool, read this very report and watch an informative video about the tool and the team. On this homepage, we have a button linking to the tool. This button will redirect the user to the file selection page where users can select pre-uploaded data sets or upload their own data set. When users choose a data set (either by selecting one or uploading one), they will be redirected to the pre-filtering page where users can apply constraints to the data that is used by the visualizations and the data set will be sent to the back-end. After applying a filter, the user gets redirected to the visualization page, which will show the visualizations. The back-end will process the data and generate the visualizations, which get pushed to the front-end as soon as they are ready.

Whilst the user is waiting for the back-end to generate the visualizations, the page shows a loading screen to indicate that the data is being processed instead of displaying a static page. This loading screen shows a node-link diagram, with moving nodes of various sizes and colours. Whenever two nodes are close enough to each other a link will appear between them. For the adjacency matrix, we also have a loading screen that shows squares of various sizes and colours bouncing around. These loading screens are fully programmed in JavaScript and disappear whenever the back-end has finished generating the visualizations, which get shown instead.

5 Pre-processing

Uploading the data set may seem like a trivial task for our tool, however, all of the data must be processed and adjusted for proper usage by NetworkX and Bokeh. Every file must be processed on upload and stored in such a way that it is easy for the data to be loaded another time, without starting the processing of the data all over again. The specific input format helps with that task, as our tool expects the data set to be a comma-separated values (CSV) file, containing graph data in the adjacency matrix representation. Whether this graph is weighted, unweighted, directed or undirected does not matter and gets detected automatically.

Upon uploading a file, it is sent to the back-end. In the back-end, basic checks will take place to make sure the file can actually be processed. These basic checks include a *file format* check and a *file integrity* check. The file format checks whether the uploaded file is a CSV file. The file integrity check will verify that the whole file has been submitted and that it has not been cut off due to a timeout. If the file passes these basic checks, it will be taken through the rest of the data processing process. If the file fails one of the basic checks however, the user is taken back to the selection page. However, supplying the graph in the wrong representation will prove to be detrimental to our tool. This is because our tool currently assumes all parsed data to be in this representation and it is *not* checked if that is the case.

After the uploading part, the back-end will read the file and convert it to a Pandas DataFrame. Due to how the data is supplied, the first column of the DataFrame would get the label ”unnamed.” To correct this, each subsequent column’s label is shifted to the left such that the data in each column remains unchanged but the labels are in the correct spot again. Afterwards, the DataFrame is stored on the server in the HDF5 file format and is then subjected to filtering, as defined by the user, without modifying the saved dataset.

For the filtering process, we have two methods in place, filtering by edge-weight and filtering by degree. Filtering on edge-weights means that the user defines an interval of allowed edge weights, and all edges that fall outside of this interval are filtered out. There are two options present that determine what is done with the filtered-out edges, one option will delete the edges and if a vertex has no edges left then the vertex is deleted as well. The other option deletes vertices that have no edges that fall within the interval, but the edges themselves are not deleted. This means that if a node has multiple edges whereof some fall outside the interval and some fall inside, that all of these edges are shown. This only ensures that the nodes with no edges inside the interval are filtered out. For the filtering, by degree, the user again specifies an interval, and all nodes that have a degree that lies outside the interval are deleted.

When the filtering process has been completed, the visualizations will be generated based on the filtered data. When the visualizations have been generated, the browser will receive the Python code converted into JavaScript code to draw the data set. This Python code is converted by the Bokeh server, which also allows for callbacks. Thus in the event that data is changed or needs to change (by for instance interacting with it) a Python callback is made to alter the visualization. The browser will then draw the data and allow the user to interact with it. What is drawn depends on the type of visualization the user asked to see.

6 The Graphion Tool

The tool itself is built out of four components, the website with all the information about the team, report and video, data set selection, data filtering which we also refer to as pre-filtering and the visualizations. Each component has its own section where they will be explained in much more detail.

6.1 The Website

As anyone types in the URL in the web browser, they will land on the homepage of the website. Here the users have the option to immediately launch our tool, which will redirect them to the selection page or to have a look at the report, video and the team. The report section features an abstract, and an image of the first page of the report. Underneath the abstract is a button to download the report. The team section contains portraits of all the team members, with their respective names underneath the portraits and their title. This title is not a serious title, but they represent to some extend the tasks that each team member has had. The video section of the page contains a button that will enable the video to be played on the page itself. Instead of having the video embedded by default, we have opted to hide it behind a button press as this makes the web-page interactive.

6.2 The Selection Page

When the tool is launched, the user lands on the selection page as depicted in Figure 2. On this page, the user can choose whether to upload their own data set or pick one from the list of previously uploaded data sets. If the user wants to upload their own data set, they should press the ”Upload File” button, select the CSV file and then click the ”Submit” button. If the user wants to pick a previously uploaded data set, one should click on the name of the data set. This list contains the 10 most recently uploaded data sets. After pressing either the ”Submit” button or on the name of a previously uploaded data set, the user gets directed to the pre-filter page.

6.3 The Pre-Filter

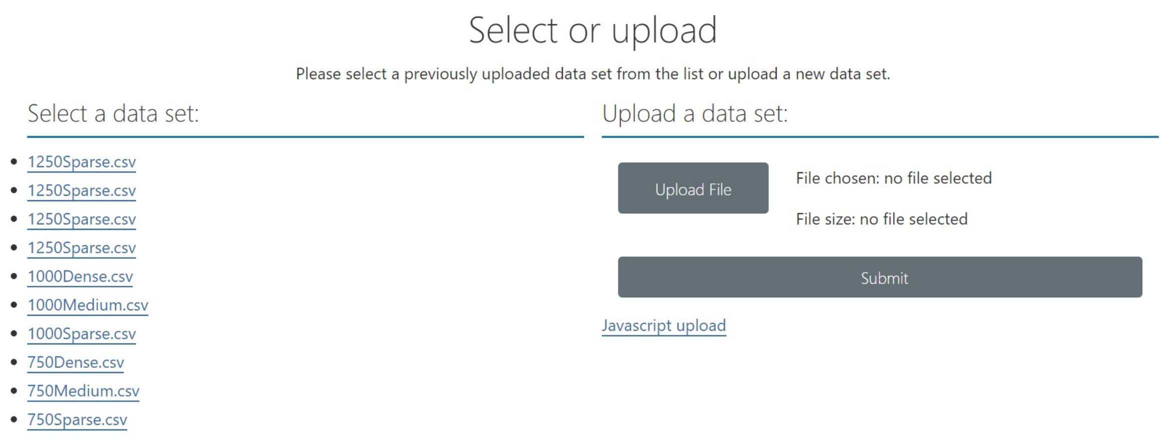
The pre-filter page shows the user an easy to use interface to filter out certain parts of the data. There are two buttons present, and these buttons describe the intended use of the filter. If the user is only interested in edges of a certain weight, then the user presses the edge-weight button. However, if the user is more interested in vertices of a certain degree, the degree-filter button is the one that should be pressed. Combining both is also an option, then the user needs to press both buttons before continuing. After pressing the continue button, the user gets into a second menu where an interactive distribution plot is present, to help the user filter the data. Within this plot, the user defines an interval where either the edge-weight and/or degree must adhere to. Underneath the plot is a number that tells the user how many vertices will be shown with the current filter, and this number’s colour gives the user a time indication. If the colour is reddish, then the rendering process will take quite some time, if it is yellowish it will take a medium amount of time and if it is greenish it will be done quite fast. When the user has specified the filter to be applied, the button continue to visualization must be pressed to start the filtering and rendering process, whereupon the user is redirected to the visualizations page.

6.4 The Visualizations

Our tool supports two different visualizations, a node-link diagram and an adjacency matrix. These two visualizations are displayed next to each other on the same page so that one can easily compare the two views. Both visualizations support hovering, where one can hover over an object (being a node or link) to obtain more information about it.

6.4.1 Node-link Diagram

The node-link diagram is the most commonly used visual representation of a graph. It consists of a set of nodes, which are usually depicted by small circles or ovals and a set of links, denoted by lines. Links can only exist between two nodes and may either be directed or undirected. A graph can only contain one type of links and is thus as a whole directed or undirected. The links that are directed, are depicted by arrows where



Select or upload

Please select a previously uploaded data set from the list or upload a new data set.

Select a data set:

- 1250Sparse.csv
- 1250Sparse.csv
- 1250Sparse.csv
- 1250Sparse.csv
- 1000Dense.csv
- 1000Medium.csv
- 1000Sparse.csv
- 750Dense.csv
- 750Medium.csv
- 750Sparse.csv

Upload a data set:

Fig. 2. The user interface of the selection page. The Javascript upload button is in the testing phase and is therefore it is not described

the arrowhead indicates the direction of the link. The layout and order of the nodes in a node-link diagram can differ quite a lot.

Initially, all nodes will be placed on a circle and spread evenly and all links will be shown between the nodes, whether the supplied graph is directed or not determines what type of link is used. The initial order of the nodes on the circle is arbitrary. However, there are other layouts available, like force-directed or hierarchical. The order may also differ, like alphabetically sorted or sorted by degree (the number of links connected to the node). See figure 3 for an example of a radial node-link diagram of a small undirected graph.

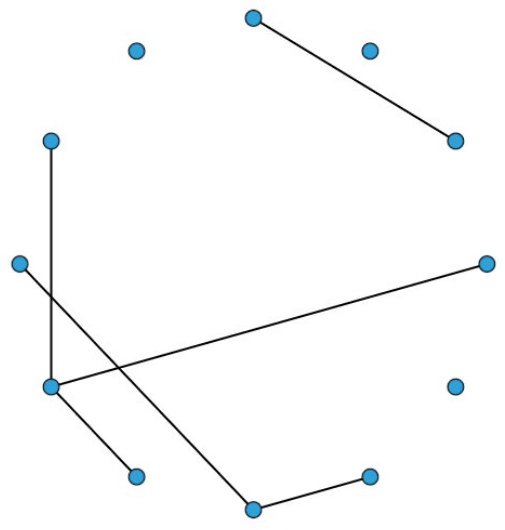


Fig. 3. Example of a radial node-link diagram

If the provided data set contains 2500 edges, the entire interior of the circle will be black because too many links are drawn. To solve this, the user can interact with the graph. After clicking on the zoom icon on the right of the graph, the user can zoom in by using the scroll wheel of the mouse. By zooming in close enough, it will be possible to distinguish the individual links from each other. Another method to distinguish individual links is to hover over a node. By hovering over a node, the node and its connected links are highlighted in green and a pop-up displays information about the node, like the label of the node. By clicking on this node, the node stays visible and all other nodes fade away to the background (however they will still be faintly visible).

6.4.2 Adjacency Matrix

The adjacency matrix is another powerful visualization to analyze data. An adjacency matrix is a matrix (or table in simple words), where each slot in the matrix is a number that denotes how strong the relationship (if any) between two nodes is. Then all numbers in the slots are mapped to colours (in our case, black for low numbers and blue for high numbers by default). This way, the matrix/table is basically translated to an image where each slot (which can become as small as a pixel for large data sets) displays a number in the original matrix. Figure 4 shows an example of an adjacency matrix.

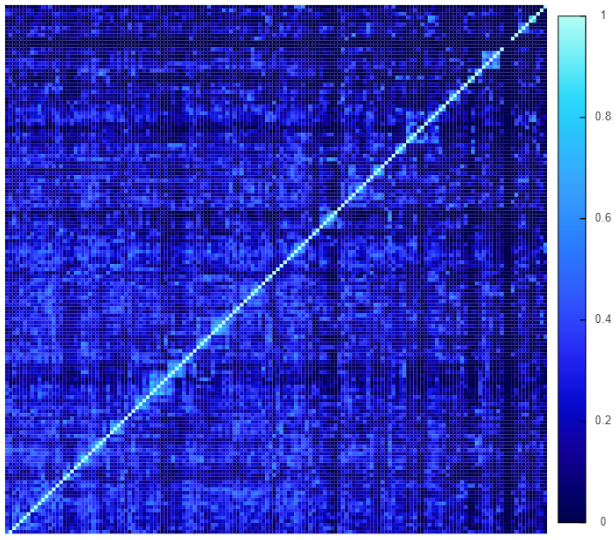


Fig. 4. Example of an adjacency matrix

Notice that the adjacency matrix above has already been ordered before it was drawn. Due to the ordering, the colour of a slot is usually quite similar to the colour of its neighbour slots. If it would not have been ordered, slots would often have random colours and the plot would be very hard to read. That is why finding a proper order for the rows and columns composing a visual matrix is essential for understanding data sets. This random order is usually the case when the matrix first shows up, after uploading or selecting a data set, as the initial matrix preserves the ordering that is present in the data set. Hereby we assume that the data set contains unordered data. If the data in the data set is ordered, this order will be shown in the initial matrix and can be retrieved later.

In our tool, we use 7 re-ordering methods with different ways of calculating the similarity between two clusters in various distance metrics.

6.4.3 Matrix Re-ordering Algorithms

All of our adjacency matrix re-ordering ways are based on the hierarchical tree. The algorithm we use is provided by the python package *fast-clustering*. Since the input dataset is a directed weighted graph, the adjacency matrix would be either symmetric or asymmetric, but always with $n \times n$ size, where n is the number of vertices of the graph. The general procedure of our method works in three main steps:

- Generate a symmetric distance matrix based on the dataset the user uploaded.
- Calculate a hierarchical tree with hierarchical cluster analysis using different clustering methods, provided by the fast-clustering package.
- With the obtained hierarchical tree, traverse all the vertices from root to leaves.

The agglomerative clustering method applied on an adjacency matrix and can be summarized as follows:

First, each node is treated as a single cluster, i.e., there are n clusters for n inputs nodes. Then, among all pairwise weight edges, a pair of nodes with shortest distance is clustered together and converted to a new node while removing themselves from graph. After that, we have $n-1$ nodes, and the distance from the new generated node to all the other nodes are determined by 7 methods which are listed below. The whole procedure would be repeated $n-1$ times until all nodes are clustered into a single node.

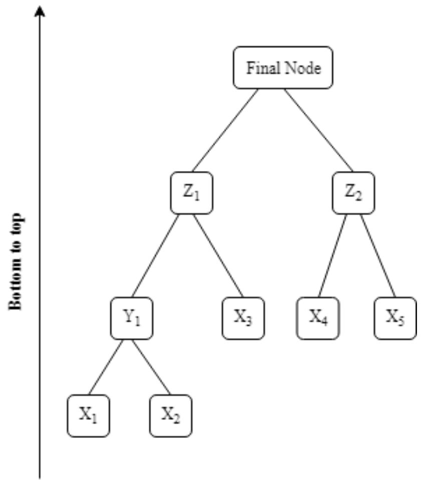


Fig. 5. Agglomerative Clustering Method: From single nodes to a whole cluster

- Single: The minimum distance between any nodes in two clusters
- Complete: The maximum distance between any nodes in two clusters
- Average: The average of summation of distance between all nodes in two clusters.
- Weighted: The average of the two distances determined by previous clusters.
- Centroid: The distance between centroids or means of two clusters.
- Ward's Method: The ESS (Error Sum of Squares) of the new cluster after combination of previous two clusters.
- Median(Weighted Pair Group Method using Centroids): The distance between centroids including counting weights of two clusters

The first three methods are graphics-based since the new clusters are presented by previous clusters or original sample points. The last four methods are geometric-based because the distances of new clusters are determined by a center point of previous clusters.

There are some shortages of the single-link and complete-link [1] methods, which can be improved by the average-link method. For the single-link method, it is easy to end up with a long chain and it causes clusters to spread out due to its clustering mechanism: only two nodes in the whole graph need to be close enough to form a pair, all others are irrelevant. The complete-link suffers from "crowding", since it only considers the worst-case distance between clusters. In other words, the node can be closer to the nodes in other clusters than its own. Average-link, which avoids both chaining and crowding, may change with a monotone increasing transformation of dissimilarities. Although none of these three clustering methods are optimized, users can choose their preference according to their demands.

Besides clustering methods, we also provide 11 different distance measures (metrics) to users. The default option is set to *Euclidean metric*, which measures the *ordinary* distance (as the crow flies) between two points in the Euclidean space. In most cases, geometry-based' clustering would use *Euclidean metric*. Another commonly used metric is *Cityblock metric*, computing the Manhattan distance between the points, normally applied in data sets as networks.

Out of these re-orderings, users can choose all options that they like and the matrix will be reordered afterwards. The drop-down menu allows users to choose several different methods described above to rearrange their data sets. This way, users can try all ordering methods to find information or patterns they are looking for. Also, the user can move the matrix around, by dragging the matrix to a different place within the visualization and can zoom in and out with their scroll wheel.

References

[1] E. Althaus, A. Hildebrandt, and A. K. Hildebrandt. A greedy algorithm for hierarchical complete linkage clustering. In *International Conference on Algorithms for Computational Biology*, pp. 25–34. Springer, 2014.

[2] B. Bollobás. *Modern Graph Theory*. Springer, Berlin, Heidelberg, 1998. doi: 10.1007/978-1-4612-0619-4

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 3rd ed., 2009.

[4] R. Diestel. *Graph Theory*. Springer, Berlin, Heidelberg, 5th ed., 2017. doi: 10.1007/978-3-662-53622-3_1

[5] M. Ghoniem, J. . Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *IEEE Symposium on Information Visualization*, pp. 17–24, Oct 2004. doi: 10.1109/INFVIS.2004.1

[6] J. Matoušek and J. Nešetřil. *Invitation to Discrete Mathematics*. Oxford University Press, Great Clarendon Street, Oxford OX2 6DP, 2nd ed., 2009.

[7] M. Okoe, R. Jianu, and S. G. Kobourov. Node-link or adjacency matrices: Old question, new insights. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018. doi: 10.1109/TVCG.2018.2865940

[8] D. Vogel, G. Dickson, and J. Lehman. *Persuasion and the Role of Visual Presentation Support: The UM/3M Study*. University of Minnesota, Minneapolis, Minnesota 55455, June 1986.

[9] 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):1224–1231, Nov 2007. doi: 10.1109/TVCG.2007.70515