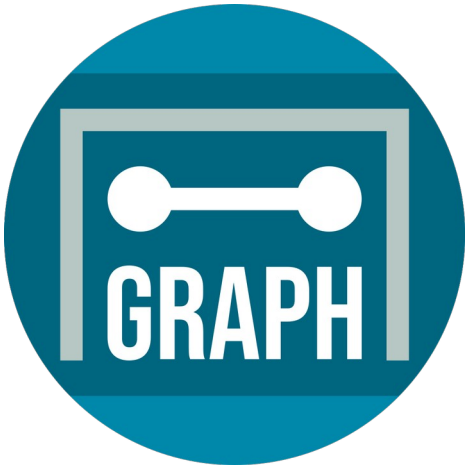
UCLouvain – LDATA2010 Information visualisation

Implementation of a network visualisation software



AGraph (User guide)

Lucien Ledune & Oscar Liessens

December 2019

AGraph is a lightweight, user-friendly and efficient graph viewer. Its purpose is to allow the user to interact with their data in a smart way using a simple interface, providing them with a rich overview of the network they're working with. In this user guide, we will detail the structure and the different possibilities offered by AGraph.

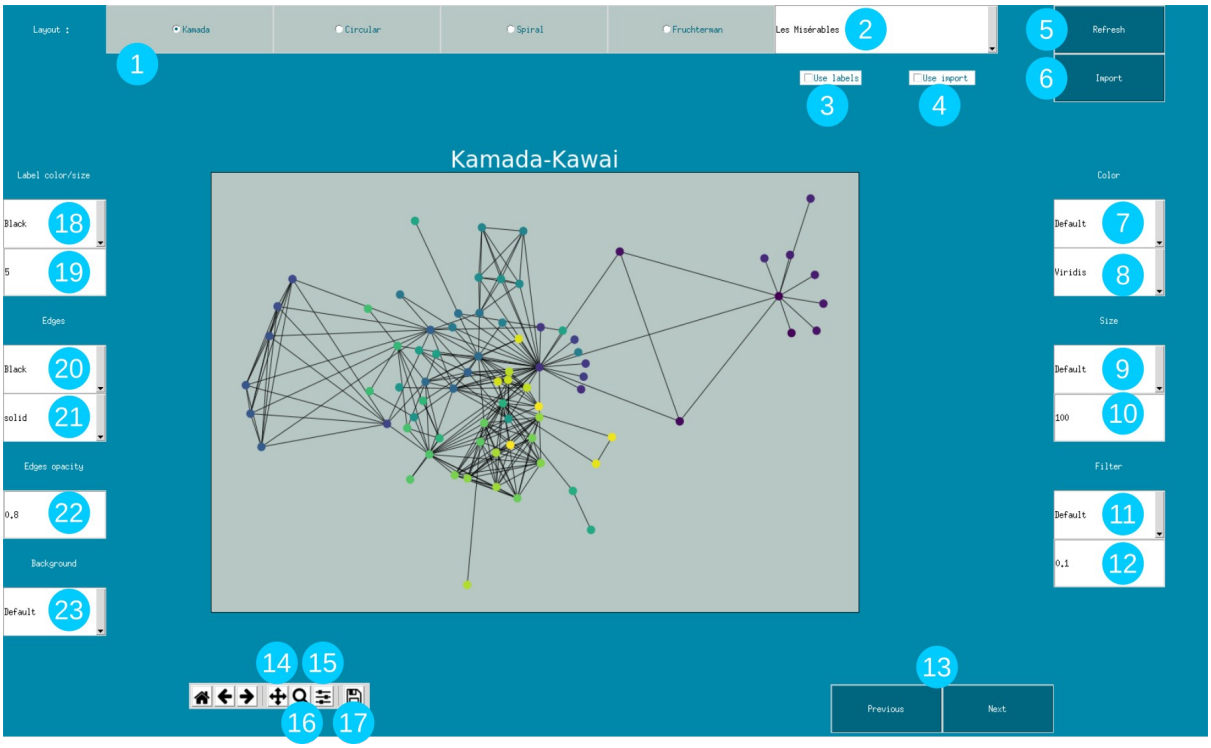
(1) Introduction

Back-end visualisation tools. AGraph is a Python software that uses several existing packages in order to provide network visualisation. Importing graph data and providing coordinates for the layout of the nodes is done by the *networkx* package (Hagberg et al, 2008). This package is used as well for computing the metrics of the nodes, except for the displaying of communities which relies on the *community* package (Aynaud, 2018), implementing the Louvain algorithm (Blondel et al, 2008). AGraph also uses *matplotlib* (Hunter, 2007) in order to generate the actual representations of the network. Finally, the user interface system is provided by the *tkinter* package (Shipman, 2013), and we used the *palettable* package to provide satisfying colour choices for the visualisations.

General functioning. In order to allow the user to tweak several settings at once without making their system reload the visualisation every time, we have made the conscious choice of not implementing changes in real time. Instead, whenever the

user is satisfied with the current settings, the button “Refresh” [5 ; see Overview of the user interface] has to be pressed in order to make the changes appear. This holds true for every functionality presented hereafter, and will not be reminded when presenting them.

Overview of the user interface.



(2) Importing data

Preloaded datasets. AGraph comes with several free datasets built-in, so that the user may immediately discover its functionalities. Namely, these datasets are:

- **Zachary Karate Club :** Social network among 34 members of a university karate club, originally used to design an anthropological model of information flow (Zachary, 1977)
- **Les Misérables :** Map of co-appearances between the characters of the novel *Les Misérables* by Victor Hugo (Knuth, 1993)
- **Airlines :** Network of airport connections packaged by Gephi (Bastian, Heymann & Jacomy, 2009), of unknown origin.

In order to switch to a different preloaded dataset (Les Misérables is loaded by default), the user has to use the dedicated drop-down menu [2] and choose the one they want.

Importing new datasets. AGraph also allows the user to import their own dataset, in either GEXF or GML format. In order to do so, one first has to check the “Use import”

button on the top [4] .Then, using the “Import” button [6], the user can select a file with an allowed extension.

(3) Navigating visualisations

Pan. Clicking this icon [14] allows the user to move the network view vertically and horizontally by holding left click on the graph and moving their mouse around. If the user move their mouse around while holding right click, they will be able to resize the network on the horizontal and vertical axis.

Zoom. Clicking this icon [16] allows the user to zoom in on a part of the network. Once the button is clicked, they must draw a rectangle on the view by holding left click on the view. In order to zoom out, the same has to be done but while holding the right click.

Configure. Clicking this icon [15] allows the user to resize the canvas on which the graph is shown.

Save. Clicking this icon [17] allows the user to save the present visualisation by choosing a path for the image file.

Previous and next. It is always possible to find the previous views of the network, and then to go back to the most recent one. Indeed, the user can navigate visualisations with the “Previous “and “Next” buttons [13].

(4) Interface options

Layouts. In order to allow the user to view their network in multiple ways, thus gaining better insight of its structure, we have implemented the possibility to use four different layout algorithms. These will determine where the nodes are projected on the canvas, and in different cases some layout may better suit your needs than another. The implemented algorithms are:

- **Kamada-Kawai :** This is a simple layout algorithm for drawing undirected and weighted graphs (Kamada & Kawai, 1989). One of the first algorithm to address this problem, it relies on the idea that the balance of a graph, i.e. the showing of structures in the most symmetric way, matters at least as much as the reducing of edge crossings in order to improve human understanding. It incorporates the idea that edges act like springs and, at the same time, bind an repulse the nodes.
- **Circular :** A rudimentary layout option, where the nodes are placed in a circle without minimizing edge crossings.

- **Spiral :** A rudimentary layout option, where the nodes are placed in the shape of a spiral.
- **Fruchterman-Reingold :** As a later take on the model of springs of Eades (1984), Fruchterman and Reingold (1991) proposed a force-directed algorithm for the laying out of graphs. By analogy with physical systems, the nodes are placed by finding an equilibrium in different simulated forces acting on them. One of the heuristics used by the authors is to strive for uniform edge lengths. The main difference with the Kamada-Kawai algorithm (1989) is that there is a risk of local minima, because of random starting points for the nodes. In practice, the implementation we used will provide a different layout each time the visualisation is refreshed.

In order to switch layout algorithm, the user can choose from our selection by clicking one of the dedicated radio buttons [1].

Interface options. In order to better suit the user's needs and sensibilities, we provide different choices for the interface. These parameters do not reflect any node metrics. The user can therefore modify the colour of the edges [20], their appearance [21] or their opacity [22] via an entry box. The colour of the visualisation background can also be modified via a drop-down menu [23].

Node colours. It is possible to brighten the visualisation with various colours from ranges provided by *palettable*. To choose a set of colours, the user must select a palette from the relevant drop-down dialogue [8]. The colours will then be applied, according to the metrics from the menu on top [7]. When "Default" is selected, the colours will simply reflect the labels of the nodes and may be considered random.

Node sizes. The size of the nodes can be modified by two means. In order to make some nodes nigger than others based on some metrics, the user can select the relevant metric in a drop-down menu [9]. Whether a metric is selected or not, the user can also directly affect the size of all nodes by changing a multiplying constant [10] that will a proportional effect over the network.

Filter. Does the graph have too many nodes? The user can select a node metric [11] and a minimum value [12] for nodes to be shown.

Labels. It is always possible to show the nodes labels by clicking on the "Use labels" button [3]. The colour of the labels are modifiable in a drop-down menu [18], as is their size via an entry box [19] where the user can enter their preferred quantity.

(5) Node metrics

Communities. Finding communities is one of the main issues in network analysis. AGraph provides a way of visualizing communities thanks to the the Louvain

algorithm (Blondel et al, 2008), implemented in the *community* package to provide fast clustering of the nodes in an optimal number of communities. It relies primarily on modularity optimization, i.e. maximizing the difference between the ratio of edges in a group and the expected ratio if edges were random. Communities may only be shown via colours, by choosing “Communities” in the dedicated drop-down menu [7]. It is to be noted that the choice of colours [8] will change once the user has selected this option, and that two palettes named “Pale” and “Bright” will be available with discrete colours selections to reflect the communities.

Centrality. This corresponds to the measure of the importance of a given node. As it can be defined in several ways, we have included a few different options to display centrality.

- **Degree :** The amount of edges a given node has, reflecting its connectedness to the other nodes.
- **Betweenness centrality :** The number of times a given node is included in the shortest path between two other nodes (Freeman, 1977)
- **Subgraph centrality :** The sum of weighted walks of all lengths starting and ending at a given node. The weights decrease with path length (Estrada & Rodriguez-Velazquez, 2005)

Any of these three measures can be selected to colour nodes [7], to vary their sizes [9] or to filter them [11].

(6) The future of AGraph

AGraph is still in development, and its creators find it important to point out some current flaws that will deserve fixing in the future.

Colour mapping of communities. For now, the colours representing the communities are limited. As we rely on the *community* (Aynaud, 2018) implementation of the Louvain algorithm (Blondel et al, 2008), we have experienced some trouble fine-tuning the amount of communities we eventually get and therefore providing a large enough array of colours. In our current implementation, whenever the optimal amount of communities exceeds the available colours, different communities will be mapped to the same colour.

Manual positioning of nodes. AGraph doesn’t currently support the manual dragging of nodes in order to make adjustments to the layout. As our plan is to allow the user to interact in the most natural way with their networks, the creators will try to implement this.

User interface overhaul. The current interface of AGraph is somewhat rudimentary, as we have placed more importance on the back-end workings until now. Our plan

for the future includes the transformation of the design in order to make it more aesthetically pleasing and intuitive, along the well known Google material design guidelines.

(7) References

Aynaud, T. (2018). Community detection for NetworkX's documentation. https://python-louvain.readthedocs.io/en/latest/index.html

Bastian, M., Heymann, S., & Jacomy, M. (2009, March). Gephi: an open source software for exploring and manipulating networks. In *Third international AAAI conference on weblogs and social media*.

Blondel, V. D., Guillaume, J. L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, *2008*(10), P10008.

Eades, P. (1984). A heuristic for graph drawing. *Congressus numerantium*, *42*, 149-160.

Estrada, E., & Rodriguez-Velazquez, J. A. (2005). Subgraph centrality in complex networks. *Physical Review E*, *71*(5), 056103.

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, 35-41.

Fruchterman, T. M., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and experience*, *21*(11), 1129-1164.

Hagberg, A., Swart, P., & S Chult, D. (2008). *Exploring network structure, dynamics, and function using NetworkX* (No. LA-UR-08-05495; LA-UR-08-5495). Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, *9*(3), 90.

Kamada, T., & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information processing letters*, *31*(1), 7-15.

Knuth, D. E. (1993). *The Stanford GraphBase: a platform for combinatorial computing* (pp. 74-87). New York: AcM Press.

Shipman, J. W. (2013). Tkinter 8.4 reference: a GUI for Python. *New Mexico Tech Computer Center*.

Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of anthropological research*, *33*(4), 452-473.