The Journal of Open Source Software

The Plebian Graph Library: A WebGL based network visualisation and diagnostics package

Indrajeet Haldar [1]

1 Graduate School of Design, Harvard University, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Given a large network (greater than one million nodes), visualising and diagnosing network data has often proven challenging (Nowogrodzki, 2015). Although there is a wide range of statistical tools to draw inferences, the esoteric nature of the statistical analysis of networks limits the communication of the findings to researchers familiar with these research methods (Tobi & Kampen, 2018). Moreover, the more abstract nature of statistical solutions reduces the complexity of diagnosing network properties, especially network diffusion. It allows for limited documentation and analysis only from the perspective of mathematical models of this particular property of large-scale networks (Gao & Guan, 2012). The Plebian Graph Library (PGL) is a library that solves for the visualisation of large networks and their diagnostic study.

The Plebian Graph Library (PGL) is a javascript library (written in Typescript (Bierman et al., 2014)) designed to facilitate the visualisation and diagnostic analysis of large-scale network data in browsers using WebGL, using a backend provided by ThreeJS (Danchilla, 2012). Whether dealing with local datasets or data retrieved from online sources (APIs), PGL provides a versatile platform for conducting extensive network simulations, physical modelling, and visualisations whilst offering a range of diagnostic tools for organising network data using standard search algorithms (Mattson et al., 2013) such as network diffusions, Breadth-first search, Depth-first search and Dijkstra's search algorithm. With a rich set of diagnostic features, including network condensation, weighted edge pruning in highly connected graphs, and support for visualisation techniques like Kamada Kawai layouts (Kamada & Kawai, 1989), Hierarchical plots, Hive plots and Edge bundling (Bourqui et al., 2016), PGL empowers researchers to gain valuable insights from complex network structures. Additionally, PGL contains canonical example of the Zackary's Karate Club (ZKC) dataset (Zachary, 1977), and the Erdosh Reyni Random Graph model (Li, 2021) as a generator to study and compare network structures.

An illustrative case for the package is to diagnose large-scale network diffusion. Visualising a clustered network in 3D, where, for example, the network nodes are displaced vertically according to their recursive importance, i.e. eigenvector centralities (Lacobucci et al., 2017). A diffusion simulation is then run, and insights and diagnostics of diffusion sequences are gathered. For example, whether diffusion first occurs between high eigenvector centrality nodes across clusters or does the spread appears in the groups before spreading to other clusters, allowing for the visual study of the strength of weak ties behaviour (Granovetter, 1973). Exploratory research, analysis, communication and documentation of these network behaviours, as mentioned above, would have been complex in a traditional visualisation library where the emphasis lies on validation instead of exploratory study and diagnostics.

Statement of need

The Plebian Graph Library (PGL) addresses several critical needs in large-scale graph data visualisation. Existing software solutions for visualising large datasets are limited to local machine installations, for example, Gephi (Bastian et al., 2009), which restricts accessibility and compatibility across various devices. Additionally, those software libraries that are browser-based cater to smaller datasets and lack the scalability to analyse complex network structures as they rely on Scalable Vector Graphics (SVG), for example, Vis.JS (V., 2021), imposing performance limitations and two-dimensional visualisations. Moreover, from the perspective of visualisations and diagnosis, two-dimensional plots of large-scale graphs often result in visual noise and information loss. PGL fills this gap by providing a browser-based solution that leverages WebGL (using the ThreeJS Library). By utilising ThreeJS, PGL surpasses the limitations of traditional two-dimensional representations and SVG graphics, enabling a more comprehensive range of data stratification methods and facilitating more immersive and interactive visualisations. Furthermore, the capacity to navigate information-dense networks in three dimensions allows for greater number of nodes and a significant reduction in visual noise while diagnosing large-scale networks. Since its creation, it has enabled a primary academic study of large-scale social networks in the form of an academic thesis, On the Mathematics of Memetics (Haldar, 2022), where it was used to generate the primary inferences of the study.

Usage

Existing network libraries like NetworkX (Hagberg et al., 2008) dictated the semantics of the graph library and borrowed some of the semantic ideas from ThreeJS. The overall structure is to define a Graph Object made of nodes and edges. Then, modify this graph based on some properties and update the relevant settings. Lastly, visualise the nodes as point clouds, boxes or cylinders, and draw out the edges (bundled or not). The following is a short example of the canonical ZKC dataset visualised in the library, simulated with Edge bundling.

```
// import the library
import * as PGL from "../Build/pgl_module.js";

// construct a simple ZKC graph
// this graph is pre initialized (Since it is simulated in this case)
const zkcSimulated = await PGL.SampleData.LoadZKCSimulated();
// console log this to see how this data is stored
zkcSimulated.printData();

// set the width and height
const width = 800;
const heigth = 700;

// pass in the graph and the canvas into the drawing object to draw it
// first get the canvas element
const canvas = document.getElementById("displayCanvas");
// then create a graph drawer object
// to do that first make a options object
const graphDrawerOptions = {
    graph: zkcSimulated,
    width: width,
    height: heigth,
    canvas: canvas
};
// then create the visualization window
// with those settings
const graph3d = new PGL.GraphDrawer.GraphDrawer3d(graphDrawerOptions);
// initialize this object before adding things to it
await graph3d.init();
```

```
// Create the 3d elements for the graph
// first describe a global scaling factor
const bounds = 1
// first create all the node elements
const nodeVisualElements = PGL.ThreeWrapper.DrawTHREEBoxBasedVertices(zkcSimulated, boun
// add the node elements to the scene
graph3d.addVisElement(nodeVisualElements);

const EdgeMap = zkcSimulated.get_edge_map();
const EdgeMap_copy = PGL.Drawing.DisplaceEdgeInY(EdgeMap, 20);
const newEdgeMap = await PGL.Drawing.DrawEdgeBundling(EdgeMap_copy, 50, 10);
// then create the edge elements
// these are the reuglar edges
const edgeVisualElements = PGL.ThreeWrapper.DrawThinEdgesFromEdgeMap(EdgeMap, bounds, 0x
graph3d.addVisElement(edgeVisualElements);
// these are the bundled edges
const edgeVisualElements_Bundled = PGL.ThreeWrapper.DrawThickEdgesFromEdgeMap(newEdgeMap
graph3d.addVisElement(edgeVisualElements_Bundled);

// then there are two last steps for a 3d graph
// this is done so that other 3d objects can be added in later
// since the base library is three.js all standard three js actions are possible
// now moving on to the two steps :
// make an animation function
function animate() {
    requestAnimationFrame(animate);
    graph3d.rendercall();
}

// append the graph renderer to the container
// and then drawing render calls
animate();
```

Documentation

Package documentation is available on GitHub for the package. Guides for general guides and detailed descriptors of all the functions are also included. Further documentation is available at https://www.plebiangraphlibrary.com/. Examples are available at https://www.plebiangraphlibrary.com/examples.html.

Acknowledgements

The Geometry Lab, under the Laboratory for Design Technologies at the Graduate School of Design at Harvard University, funded this project.

References

Bastian, M., Heymann, S., & Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media*.

Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. *European Conference on Object-Oriented Programming*, 257–281.

Bourqui, R., Ienco, D., Sallaberry, A., & Poncelet, P. (2016). Multilayer graph edge bundling. *2016 IEEE Pacific Visualization Symposium (PacificVis)*, 184–188. https://doi.org/10.1109/PACIFICVIS.2016.7465267

Danchilla, B. (2012). Three.js framework. *Beginning WebGL for HTML5*.

Gao, X., & Guan, J. (2012). Network model of knowledge diffusion. *Scientometrics*, *90*(3), 749–762. https://doi.org/10.1007/s11192-011-0554-z

Granovetter, M. S. (1973). The strength of weak ties. *American Journal of Sociology*, *78*(6), 1360–1380.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference (SciPy2008)*, 11–15.

Haldar, I. (2022). *On the mathematics of memetics* [Master's thesis]. Graduate School of Design, Harvard University.

Kamada, T., & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters*, *31*(1), 7–15.

Lacobucci, D., McBride, R., & Popovich, D. L. (2017). Eigenvector centrality: Illustrations supporting the utility of extracting more than one eigenvector to obtain additional insights into networks and interdependent structures. *Journal of Social Structure*, *18*(1), 1–22.

Li, J. (2021). Brief overview of graph theory: Erdos-renyi random graph model and small world phenomenon. *University of Chicago*.

Mattson, T., Bader, D., Berry, J., Buluç, A., Dongarra, J., Faloutsos, C., Feo, J., Gilbert, J., Gonzalez, J., Hendrickson, B., Kepner, J., Leiserson, C., Lumsdaine, A., Padua, D., Poole, S., Reinhardt, S., Stonebraker, M., Wallach, S., & Yoo, A. (2013). Standards for graph algorithm primitives. *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–2. https://doi.org/10.1109/HPEC.2013.6670338

Nowogrodzki, A. (2015). Eleven tips for working with large data sets. *Nature*, *527*(7576), 105–107. https://doi.org/10.1038/527105a

Tobi, H., & Kampen, J. K. (2018). Research design: The methodology for interdisciplinary research framework. *Quality & Quantity: International Journal of Methodology*, *52*(3), 1209–1225. https://doi.org/10.1007/s11135-017-0513-8

V., A. B. (2021). *Vis.js*. https://visjs.org/

Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, *33*, 452–473.