V2 - From start up to end of experiment 1

27 November 2016 20:01

Introduction

Brief Summary of Project. To be written at the end

Review of Field

The visualisation of data is a field of critical importance that many huge companies rely on in order to make predictions, improve themselves, and get an edge over competitors. Data Visualisation is, "the presentation of data in a pictorial or graphical format [which] enables decision makers to see analytics presented visually, so they can grasp difficult concepts or identify new patterns.", as defined by SAS - a world leader in the field. This clearly defines what Data Visualisation is and how it is useful.
http://www.sas.com/en_us/insights/big-data/data-visualization.html

Visualisation of data can be static or interactive, with interactive data able to give a user more information by, for example, selecting parts of the visualisation and getting more information on it, changing parameters for the visualisation and observing changes, or the moving of nodes in a network and seeing how the rest of the network responds. This can greatly improve the usefulness of the visualisation for users, with new trends or vulnerabilities becoming clearer faster through interacting with the visualisation.

Network Visualisation is a branch of Data Visualisation involving the ability to display nodes and edges in a meaningful way to a user. This can be used for a large variety of purposes, such as how information is grouped, how subsets of data interacts with other subsets, and how interconnected or isolated the data is.
https://flowingdata.com/2010/11/17/why-network-visualization-is-useful/

Visualisation of massive networks can lead to many challenges. Massive is a very vague term, but is generally considered as anything above about ten thousand <<source inserted here>>. These challenges generally present themselves in the form of computational challenges or visualisation challenges. Computational challenges are when the a system struggles to display a network visualisation due to lack of speed or performance from the CPU, RAM, backing storage or network. If a machine has a slow processor then the amount of time to create and then render the network will be increased greatly. Similarly, if the RAM is slow or if the network is big enough that it will not fit in the available amount of RAM, then the performance of the network visualisation will degrade greatly.

The other computational challenge is the ability to store the network. Whether it is stored locally on a machine or stored somewhere on a network, this relies on fast disk access, enough storage space (which, depending on the amount of data that is to be stored to be visualised, could be nearly impossible to store on one machine). Hence, when storing huge amounts of data, it will always be stored in the cloud, which means a high quality internet connection is required.

Assuming that the machine is powerful enough in all of the above ways, and is capable of visualising massive networks, the next challenge is how to present that data in any meaningful way. When visualising a few hundred or thousand nodes, it is generally fairly easy to get useful information out of the visualisation. However, when a user is faced with a few hundred thousand nodes or even several million, it can be very difficult to gain any sort of insight from the visualisation.

The goal of the project was to research techniques currently used to visualise massive networks, look into how successful they were, and then make a system that allows for the visualisation of massive networks in a browser and across a network.

Background research

Initially, research done covered both network visualisation and specifically massive network visualisation, and as well as that, it looked into software that existed which visualised networks. In regards to the visualisation of massive networks, some important techniques learnt were:

- Node grouping/bundling. This is when software uses algorithms to decide (with or without some form of user input) which nodes are very similar to each other, and upon establishing this relationship, groups them in to one node, generally visually different in some way (often by size or colour). This allows for hundreds or thousands of nodes to be grouped into one node and hence take up for less memory, processing power, bandwidth and screen space on the clients machine. A way to make this even more helpful is to show how the nodes being bundled interact with themselves, for example if they are heavily connected or all connected to just a few nodes, or if they are in a ring formation.

- Edge bundling. This is similar to node grouping but applied to edges, when several edges take very similar paths from one node (or group of nodes) to another, then the edges can be bundled into one edge. If node grouping is done then this process will have less of an impact as all of the edges will already be bundled together assuming the node grouping algorithm was successful. Like node bundling, it is often shown through the change of the size of the edge or the colour.

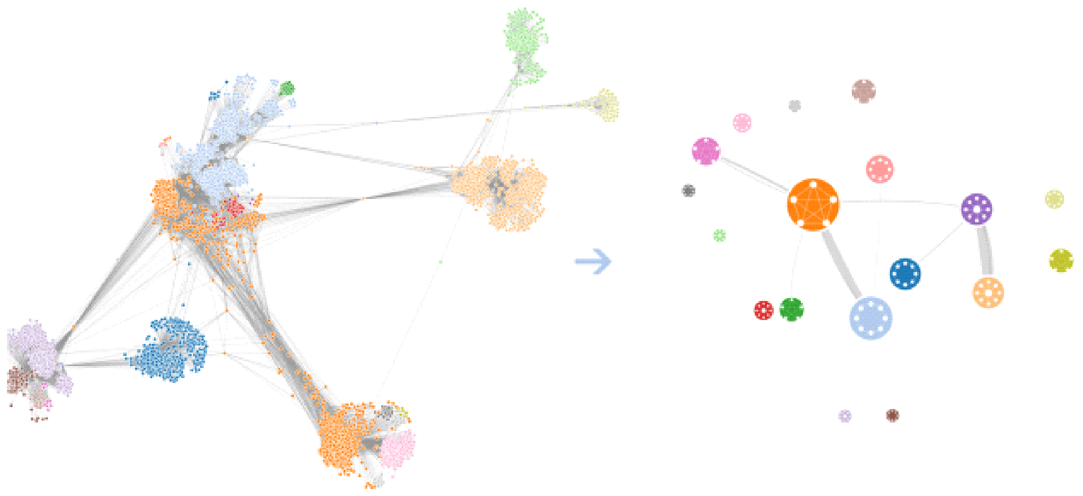


Figure X: Demonstrates both node and edge bundling, where thickness of edge displays how many edges there are, and size of node indicates number of nodes group. On top of that, each node displays how nodes within it were connected together.

- Fish-Eye-Lenses: This involves rendering all or part of the graph, and upon focusing on a section of the graph, that part is zoomed in or clarified as if through a lens and that part of the graph is zoomed in, becoming more clear. Another way of implementing this can be that when the lens is applied, it can render more information that was not shown before in order to save computational power.

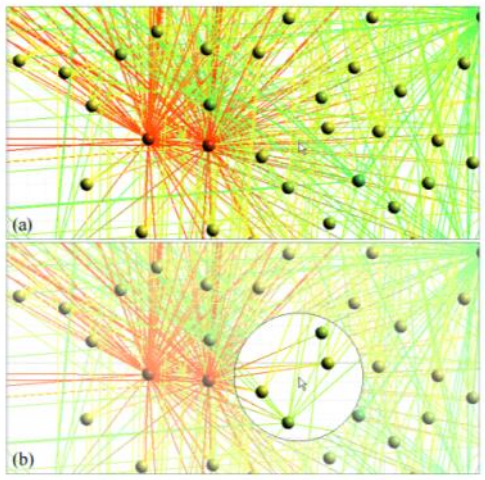
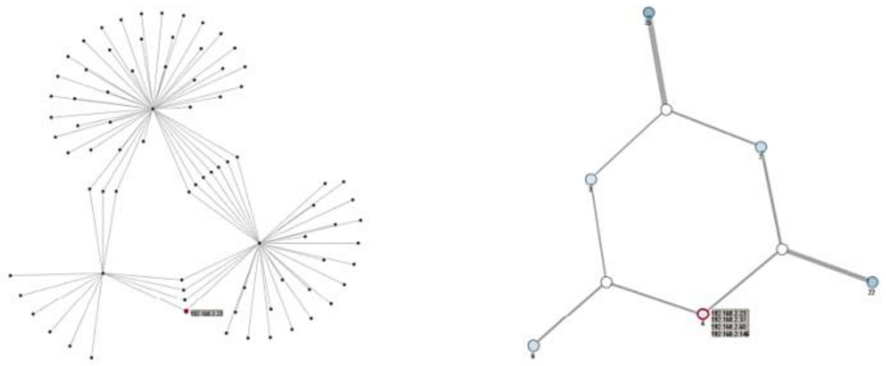


Figure 4: If graph visualizations are cluttered (a), it is hard to identify which edges are connected to particular nodes. Applying the interactive Local Edge Lens (b) enables users to accomplish this task easily. The color of an edge is interpolated between the colors of the vertices connected by the edge.

- Deleting unnecessary content "smartly": This involves using an algorithm to remove certain nodes or edges that are deemed to be adding little or no useful information for the user. This algorithm can take no parameters, or work off a user's input, in which they specify what they are interested in, so other information is partially or fully removed.



On top of finding out techniques used by software for large network visualisation, a list was also created of all software that was mentioned in papers or found during research that was related to large network visualisation. Quite often, the software mentioned had been hand-made for that project and was not feature complete, or the software could only be used for a price, but the below list contains the names of software that was reasonable to look further into, found by searching online and through mentions from papers:

- GUESS
- SNAP
- Gephi
- GraphViz
- Tulip

- D3.js
- Vis.js

Criteria and Process

<<Insert sources to back up points>>

Upon finishing background research, the next step was to come up with a list of criteria that each of the pieces of software would be compared against, and a robust process for how the software would be tested against the criteria. The process and criteria would ideally be based off an established way to systematically evaluate information visualisation. However, even after a long time was spent trying to find papers on that subject, the only papers found that were found were related to getting users to test the system as opposed to a developer evaluating existing systems.

The initial plan was to split the criteria into several categories and weigh each category as to its importance.

The first criteria to assess would be how easy it is to get a basic network (5 or 10 nodes) displayed from scratch, from downloading the software and installing it, to understanding it's documentation, to getting a network displayed.

After that, some more important criteria were how easy it is to load in a network from a file, and also, if the network was modified in any way between loading it in and displaying it to the used, or even while the user was viewing it, then saving the network to a file.

The next features to be analysed for will be based on interactivity - how easily a user can edit the network to get more useful information out of it. This would include the ability to zoom into a part of the network or being able to pan or scroll. A more complex feature (that would be rarely used but would occasionally add valuable information to the user) would be the ability to make it dynamic - showing the change in a graph over time (or another parameter).

A major criteria would then be how well the software would scale up to massive networks - this would include running tests on increasingly large networks and seeing how long the software takes, and also if it includes node or edge bundling or other ways of increasing clarity of massive networks. Time taken to render is also an important statistic, but less important than clarity for massive amounts of data for this project. This is perhaps the most important category as no matter the above features, if the network is not clear for huge numbers of nodes / edges then it does not answer the problem at hand. A feature that would be desired to increase user clarity would be the ability to view the network in many different layouts - which would give users more flexibility and would often lead to better understanding.

This set of criteria was compiled into a list which was to act as the process for the experiment.

1. Software easy to download
2. Installation necessary
3. Documentation:
   a. Easy to find
   b. Easy to understand
   c. Comprehensive

   d. Time until confident
4. Could get a network with three nodes and two edges outputted
5. For file size of 30, 200, 1000 and 3000, record how long it takes to render
6. Could load in a file
7. Could save to a file
8. Ability to:
   a. Zoom
   b. Pan
   c. Scroll
   d. Dynamic
9. Look into ability to work well with massive networks
   a. Supports:
      i. Node bundling
      ii. Edge bundling
   b. Has massive network layout options
10. Graphical properties were editable
11. Graph could be 3D
12. Multivariate
13. Record any other points of interest

Experiment 1 Notes

3.1 Aim
The goal of the first experiment was to do research into several different types of graphing software, and as a result of the research, find out more about how network visualisation software. The software would be analysed for several characteristics, both related to general graphing of networks, and graphing of specifically huge networks. A list of the criteria can be found above.

3.2 Methodology
Using the criteria, a spreadsheet was made, in which every piece of software had a different sheet, and each sheet had all of the criteria each piece of software was to be compared against. Each piece of software was then analysed (while being timed) and the results of that analysis was documented. Although effort was put in to try to make sure the analysis of each system was not biased by looking over different software beforehand, it is reasonable to assume that some of the software tested later on was easier to use as a result of learning other software beforehand.

3.3 Data / Results

Upon beginning to run the experiment, it became clear that several pieces of software would not be able to be assessed by the criteria created previously, for a number of reasons.

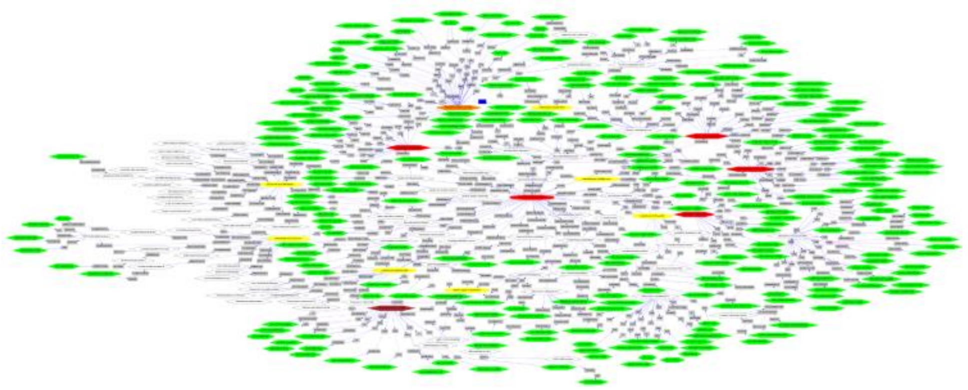
GUESS
The first piece of software, GUESS, was software that never left beta, and was last developed in 2007. This meant that the software had fairly little documentation and what it did have was poor. On top of this, the wiki no longer existed which meant many of the links to documentation on the website were broken. Despite it looking promising, it became clear it would not be useable.

SNAP

This piece of software could not be compared to much of the criteria as the purpose of it is to analyse and manipulate graphs, as opposed to also visualise them. However, time was spent reading it's documentation and going through all the sample code snippets to understanding how the software worked to some degree. The reason this was done was that it both felt important to have some knowledge of how network manipulation software worked, and also depending on the direction the project took, network manipulation may well be used in order to bundle edges or nodes in order to increase performance of massive networks.

Gephi
This was the first software package that was as expected, and was fairly easy to test, due to being recent, easy to use and well documented. After about half an hour I was comfortable with how it worked, able to make simple graphs, and to import graphs from many formats (CSV, GDF, QML, GraphML, net, etc.) and export to all of them, along with as a PNG, PDF or SVG. The time to render graphs of a few thousand nodes was less than half a second, less than a thousand nodes was instant. It supported zooming, panning and scrolling, and having data being dynamic was a possibility. There was also multiple heavily customisable layout options, some of which would suit large networks more than others, although there was no additional support for large networks, other than the system being very efficient (bundling was not supported, along with partial viewing of the data). Additionally, there was the ability to change graphical settings, have the data graphed in 3D, and have a multivariate graph.

Graphviz was difficult to test, being a collection of software as opposed to a single package. Most of the packages were not fit for purpose, often displaying data in charts, for example. The package I ended up testing was 'sfdp'. Although it fitted most of the criteria and claimed to support large networks, it turned out to be very focused on graphing far smaller networks with little support for anything of a few thousand nodes or more. The documentation was okay, and the software had the ability to import and export as a very large amount of file formats. It also supported zooming/panning/scrolling, but along with getting decreasing in speed a lot faster than other pieces of software (three thousand nodes took several seconds), the UI clearly was not build around supporting large networks, with information becoming very unclear quickly. There was also no explicit support for visualising massive networks.



http://www.graphviz.org/content/root - this shows how the UI is clearly focused towards smaller networks of up to a thousand nodes or so.

Tulip is an information visualisation framework that also lets you both visualise and analyse the data. It is very easy to use and is well documented, allows for easy importing and exporting of networks, and is really fast, allowing for 3000 nodes to be visualised instantly.

It is worth noting that on the university lab machines, without admin access, it was not possible to install Tulip, which meant I ended up installing it on Windows (the computer that Tulip was ran on is far more powerful than the university machines and there is a very reasonable chance that on the lab machines Tulip would not perform as well). However, it seemed very good quality software, performing very well for all tested sizes, and when I tried it with far bigger data sets (one hundred thousand nodes and three hundred thousand node) it still loaded nearly instantly. It also included edge bundling and 'clustering' (a type of node bundling).

D3.js is a an JavaScript library for data visualisation. As a result it is far more flexible but requires far more setup to be done by the user. The documentation is good although it is more complicated that other most of the other programs, and importing and exporting is easily possible via JSON, and other file formats with slightly greater difficulty. It supports zooming / panning / scrolling as expected and was fairly performant, displaying a few thousand nodes in about a second. Also, there are many ways to make it effective at showing massive networks, but this would require research into many different packages and potentially writing a lot code to do it.

3.4 Conclusion

In general, the software was not what was expected and hence it is hard to gauge the success of the experiment. Far more of the software was standalone packages than expected, and other issues were also come across (such as software being deprecated or for a different purpose). Also, none of the software would be directly suitable for SAS apart from D3.js as the other pieces of software were standalone applications as opposed to libraries that could be utilised by a web application.

3.5 Next Steps

As a result of the outcome of experiment one, there were several different directions that the project could go:

- Write several different programs in JavaScript or Python that alter huge graphs by bundling etc. and compare how they become easier to visualise and/or render using D3.js This would include using several large networks of different types (sparse or very interconnected) and use different algorithms or techniques to see what is the most efficient way or best way to evaluate.
- Evaluate several different types of graphing software (Tulip/Gephi/GraphViz) and see where they begin to fail and what their bottlenecks are.
- Look into database systems that visualise data too. Many database packages allow for data visualisation
- Build a web application for one of the pieces of desktop software in order to allow users to log in to a system and visualise a huge amount of data without the need for an extremely powerful machine.