Vesa-Ville Välimäki

Presenting Big Data with Interactive Data Visualization Tool

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

19 September 2017

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Vesa-Ville Välimäki Presenting Big Data with Interactive Data Visualization Tool 52 pages 19 September 2017
Degree	Master's Degree
Degree Programme	Information Technology
Instructor(s)	Heikki Lauranto, Principal Lecturer

Analyzing big data is not easy with lightweight tools. Recent data leaks have shown that exploring data with traditional methods takes too much time. This thesis investigated whether today's data visualization JavaScript libraries can create agile and interactive user interfaces. Browsers are facing hard times when massive amounts of data are analyzed on the screen.

The study was done in three parts during the year. The history of data visualization was studied first and after that the most well-known cases in the industry were investigated. Web technologies that have boosted the development were also under the research.

In the second phase, the best JavaScript libraries supporting interactive data visualization were selected. The examples provided by the libraries were studied in terms of speed, user friendliness and future development.

At the end of the studies, the best JavaScript libraries were heavily pushed to handling 100,000 elements on the screen. The survey material used was massive and open data provided by the San Francisco City and the County. This huge data file contained property information from San Francisco City area.

The study focused on the interactive side of the issue. This means how many nodes and edges visualization can take so that the user could still analyze the data smoothly. Interactivity was the key element of the study, so the drawing capability of static nodes and edges was not measured. The number of edges used in the tests was held in 500 pieces at all times. The study also marked out how the performance of the computers or the different browsers affected the results.

The final results were in line with the technical specifications JavaScript libraries promised. Today's JavaScript libraries are able to produce, together with the modern Internet browsers, a sufficient number of elements to the screen without slowing down. Best libraries are also able to offer good editing opportunities to users who lack sufficient computing skills.

Keywords	Interactive Data Visualization, Big Data, JavaScript, Open source, D3.js, Sigma.js, Linkurious, Ogma

Contents

List of Figures

Abbreviations

APGL	Affero General Public License. A free, copyleft license.
BLOG	A truncation of the expression "weblog". A discussion or informational website published on the World Wide Web.
CSS	Cascading Style Sheet. A style sheet language used for describing the presentation of a document.
D3.js	Data-Driven Documents. A JavaScript library for producing dynamic, interactive data visualizations in web browsers.
DIV	Division. Means the tag which defines a division or a section in an HTML document.
DLL	Dynamic-link library. Microsoft's implementation of the shared library concept.
DOM	Document Object Model (DOM). A cross-platform and language-independent application programming interface.
HTML5	Hypertext Markup Language fifth version. A programming language that supports the latest multimedia.
JSON	JavaScript Object Notation. An XML style open-standard file format used for communication in several data visualisation libraries.
NASA	National Aeronautics and Space Administration. An independent agency responsible for the civilian space program, aeronautics and aerospace research.
PHP	Hypertext Preprocessor. A server-side scripting language designed primarily for web development.

SVG	Scalable Vector Graphics. An XML-based vector image format for two-dimensional graphics with support for interactivity and animation.
WEBGL	Web Graphics Library is a JavaScript API for rendering 3D graphics within any compatible web browser without the use of plug-ins.
XHR	XMLHttpRequest. An API in the form of an object whose methods transfer data between a web browser and a web server.

1 Introduction

Talking about Big Data has been around so long that people are already bored, at least in the area of Information Technology where everyone knows there are lots of data out there. According to surveys, companies are already in the point where they struggle with the huge amount of the available data. (Gartner, 2016).

Many companies do not know how the Big Data should be utilized. Many of them do not know how to analyze it. Some of the companies have been so helpless with their data volume that they have had to look for unusual solutions. In order to get benefits from the data, many traditional companies have been sharing their data to public, hoping that someone will find something useful to analyze (St-Maurice, 2015).

One of the ways which data analysis can be facilitated is to use Interactive Data Visualization tools. Putting the tools into practice allows one to create images or presentations that will help public to understand the meaning of the massive data volumes. These tools are suitable for tasks where the data interconnections and dependencies are explored in different situations. If the visualization tool is compiled with quality researching can be fast and entertaining (Murray, 2013).

An interactive Data Visualization tool can be very useful for many professionals. Economic researchers can analyse the data about their customers' financial needs, healthcare professionals search leads from their patients' histories, and sound engineers are looking for a perfect acoustic environment for their recording sessions (Chun-houh Chen, 2008).

The problem is that the quality of the ready-made Interactive Data Visualization libraries has been very low so far. In most of the cases, the libraries have only been designed for little projects or small amount of data. Decreasing the data from the visualization have been the common way to solve the problem temporarily (Yuk, 2014).

Although the users can modify the libraries by themselves, the skill level required to edit them has been too high for a non-programmers. The purpose of this thesis was to search for a suitable Interactive Data Visualization engine for large data. The search focuses on

the engine which did not require any special programming skills and was working fluently with the modern Internet browsers. Nowadays most of the visualization engines are made with a JavaScript language, so they were closely under investigation.

Studies were started by exploring the world of Infographics and Data Visualization. Chapter 2 goes through the history of Data Visualization and represents other major web technologies. The same chapter also tells about the products that had been played a significant role in the development of Data Visualization. Web-based visualization tools have gone through an interesting development track over the years.

Chapter 3 is dedicated to the evaluation of four modern JavaScript libraries which were tested theoretically and practically. The performed tests measured the libraries' ability to implement a user interface which is fast, easy-to-use and behaving interactively with Big Data. Visualization examples offered by manufacturers were under the investigation too, as well as some well-known cases from around the world. There are few books about the latest JavaScript libraries, so relying on the information from Internet was a main source in this Thesis.

After searching for the JavaScript engines, Chapter 4 describes how the suitable data was searched for the tests. The mission was to find an open data file as big as possible. The one that should be adapted also to a suitable format for JavaScript engines.

Chapter 5 describes how the two most appropriate JavaScript libraries were tested. With simple performance tests, it was measured how fast data appears on the screen from web server to client. Attention was also paid to the visual appearance and the possible slowness of the examples.

Finally, in Chapter 6 a closer look was taken on the results and they are compared to the promises of the manufacturers. The discussion is on whether the existing JavaScript libraries were ready for the challenges of Big Data.

2 Interactive Data Visualization

Speaking of an Interactive Data Visualization, there is a need to go through at least two terms: Infographics and Static Data Visualization.

2.1 Infographics

One of the oldest examples of Infographics is probably the ancient cave paintings on a wall 35 000 years ago (Lankow, et al., 2012). Who would not know the figure of a man and a bison? Although the information behind the drawings might be unclear for us, it is easier to guess the message with the help of the picture. At least, easier than trying to decode the Stone Age letters.

Another familiar example of Infographics is the Egyptian hieroglyphics. This ancient style of writing were used 5000 of years ago and it contained over 1000 pictures. Nearly half of them were used in ordinary life. Because of the large amount of signs, British Egyptologist Sir Alan Gardiner made a grammar in 1927. The book was called Egyptian Grammar, and it helped people to read hieroglyphics (Roy A. Adkins, 2002).

The term Infographics has been described well in the book called The Power of Infographics:

> A visualization of data or ideas that tries to convey complex information to an audience in a manner that can be quickly consumed and easily understood (Smiciklas, 2012).

Comparing to Data Visualization, these features are characteristics for Infographics (Yuk, 2014):

- Further use of images
- Less information / text
- Clear conclusions
- More artistic
- Static

In the 20th century, Infographics were often used in newspapers and periodicals. At that time they were simply illustrated and the style was not in the center. Internet offered an

opportunity to spread the Infographics work worldwide and it started a little phenomenon. Artistically talented people were joining the Infographics scene in the beginning of the 21st century. More and more beautiful publications started to appear on the market. Figure 1 illustrates a simple infographic image.

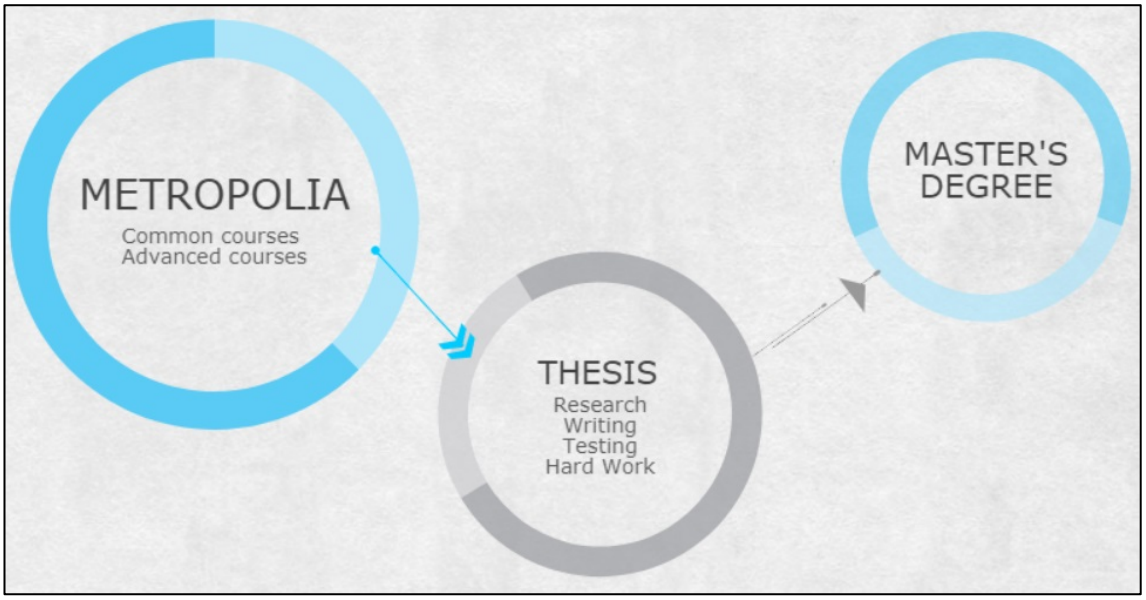


Figure 1. Simple infographics image made with easel.ly online infographics tool

After that the rise of the Infographics, editing applications began in the Internet. These tools inspired even ordinary people to make and share their Infographics over the Internet (Hohenadel, 2014). Figure 1 shows the one of those drawings made by the popular online editor easel.ly.

2.2 Static Data Visualization

Data Visualization is the way to visualize large amounts of data in a format that is easily understood by viewers. The simpler and more straightforward presentation, the more likely the viewer will understand the message (Thomas, 2015).

Data Visualization is an interesting specie. Scientists often know everything about the data itself but issues related to visualization might be hard for them. User interface designers and graphic artists dominate visual aspects, but data processing is unapproachable to them. Data Visualization enables both parties to become familiar with the unknown pieces. They surely benefit from each other that way. Data Visualization is both science and the art (Dormehl, 2014).

Comparing to Infographics, these features are characteristics for Data Visualization (Yuk, 2014):

- More numbers / figures
- More information / text
- Focused on presenting data

Today's Data Visualization has moved to a web-based interactive presentations. The increase of the data has also led more and more visual oriented presentations. Data lies in the background because of the lack of space. Visual user interfaces today are facing tough challenges, as analysis have to be done on visual terms. As a result, a growing amount of dashboard style user interfaces will appear. Those user interfaces have a visual focus and data needs to be adjusted on the fly (Koppal, 2017). Figure 2 shows an example of data visualization made with Microsoft Excel's Family Budget template.

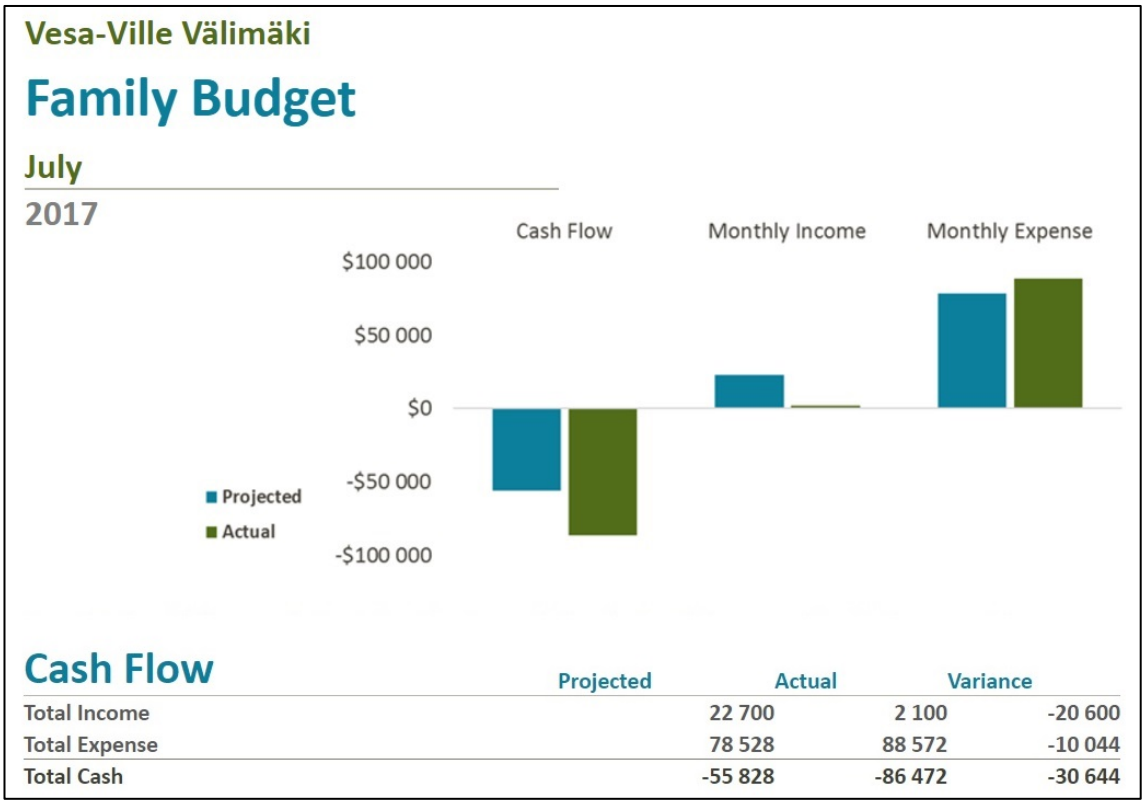


Figure 2. Data Visualization made with Microsoft Excel's Family Budget template

One of the best and the most well-known examples of a Data Visualization application is the Microsoft's Excel. The Data Visualization example in Figure 2 is based on the Excel's free template called Family Budget. Under the Cash Flow heading, the family's planned and actual cash flows are entered in the Total Income and Total Expense fields. The application calculates and automatically draws the chart bars. The ratio of the expenses and the income is available in the picture. The Cash Flow column in the chart shows whether the family's financial situation is going to be better or worse.

2.3 Evolution of Web-based Interactive Data Visualization

The first steps in the world of Interactive Data Visualization is a relatively mixed concept. According to general information, the first glimpse into interactivity began in the 1960s. Approaching the 1970s, the first interactive applications for statistics were invented. However, it still took many years before they were in general use (Friendly, 2008).

2.3.1 Prefuse

Moving to the history of web-based data visualization engines, the pioneer of this era was the tool called Prefuse as seen in Figure 3. It was published in 2005 and it was completely built with Java language. In order to run in a web browser, it was converted to a java applet that needed a separate java plugin to work. Prefuse was the first web-based interactive data visualization tool which let non-programmers also to modify the examples (Murray, 2013).

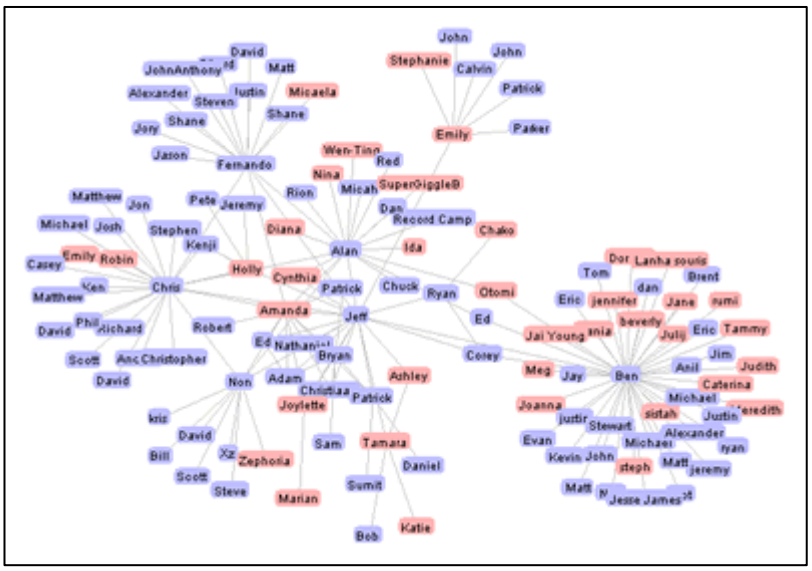


Figure 3. Picture of Prefuse example application (Sourceforge.net, 2006)

Prefuse's development with the original founders was stopped in 2011, but the story continues in the Github community. Github is a popular version control service where users across the world can develop a common project. Prefuse’s code is available free to everyone who wants to explore it.

2.3.2 Flare

One of the Prefuse's developers, Jeff Heer, released a new ActionScript based data visualization tool in 2007. It was called Flare and it required a separate Flash plug-in in order to work in the web browsers. Flash was a popular multimedia platform which was supported by almost all browsers in the beginning of 21th century. ActionScript was a programming language that was basically based on the same standards as JavaScript language. It could be almost called as a relative language. That is why Flare was a big step towards JavaScript based data visualization tools (Murray, 2013). Figure 4 shows as an example how beautiful dependency graphs Flare was capable of doing.

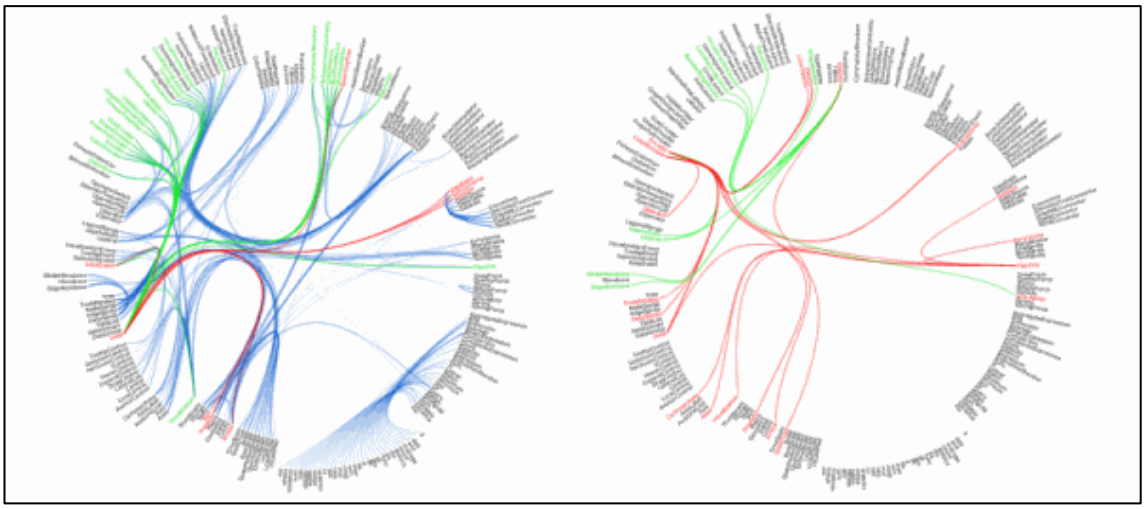


Figure 4. Flare dependency graph (Heer, 2008)

Flare was the first data visualization tool that conquered the big media companies. Images and interactive applications made with Flare were used in the news of ABC, BBC and IBM Visual Communication Lab (Lab, 2008).

2.3.3 Protovis

The next big thing among the Data Visualization tools was released in 2011. The tool was called Protovis, and Heer was again in the lead of development, this time with Michael Bostock, a student from Stanford University. Together they developed a JavaScript based visualization tool which basic principle is still in use (Murray, 2013).

The list of basic principles of working with JavaScript libraries:

- Create a HTML page
- Refer to JavaScript library at HTML code
- Call JavaScript functions
- Test HTML page in the browser

Figure 5 shows a collection of Protovis applications.

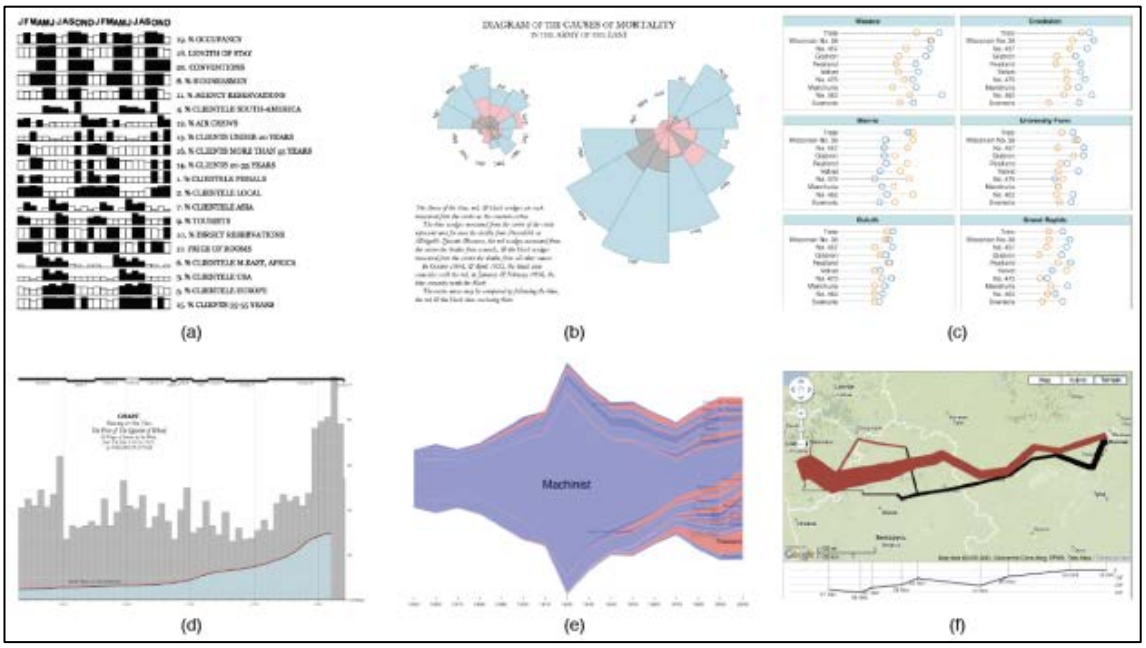


Figure 5. Protovis example applications (Michael Bostock, 2009)

The most revolutionary innovation of Protovis was that the browser did not need a separate plugin. This was a small but important step forward. The download and installation of separate plugins were commonly considered to be a difficult task. Protovis lowered the threshold so that many people began develop their own data visualizations, like in Figure 5 (Glenn J. Myatt, 2011). Figure 6 shows an example tested with Protovis.js.

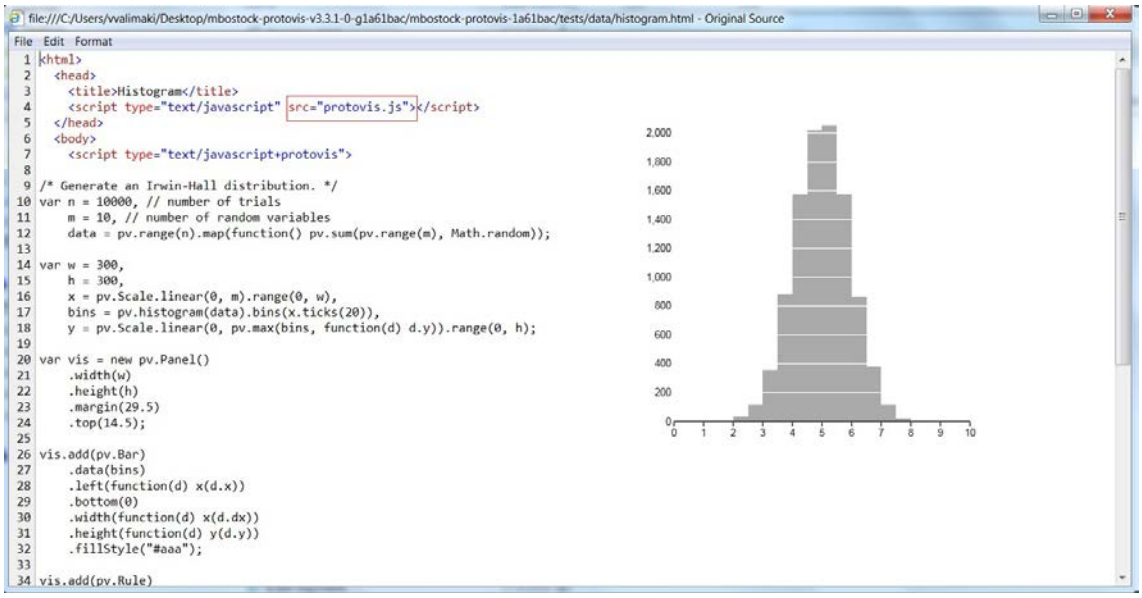


Figure 6. Histogram example tested with Protovis.js

Figure 6 shows an instance where Protovis library version 3.3.1 was downloaded and the histogram example was chosen to run. The results of this test gave a good starting point to further investigations on how easily the first JavaScript visualization libraries were ready to use.

2.4 Supporting Web Techniques

In the world of Interactive Data Visualization, some of the other web technologies will always come to picture. Those technologies have played a significant role in the data visualization development. Below are listed some of the most important ones.

2.4.1 Scalable Vector Graphics

Scalable Vector Graphics (SVG) was published in 1999 and it continues to play an important role in web development. The image format compiled in XML language can be scaled to any size. Despite the size of the image, the loading time of the image remains the same. With the help of the XML code, the image can be programmatically rotated and scaled. SVG is also popular because it can be controlled in the browser via help of the JavaScript (Campesato, 2003).

This much code is needed for a SVG image with a blue circle and a Metropolia text (Figure 7):

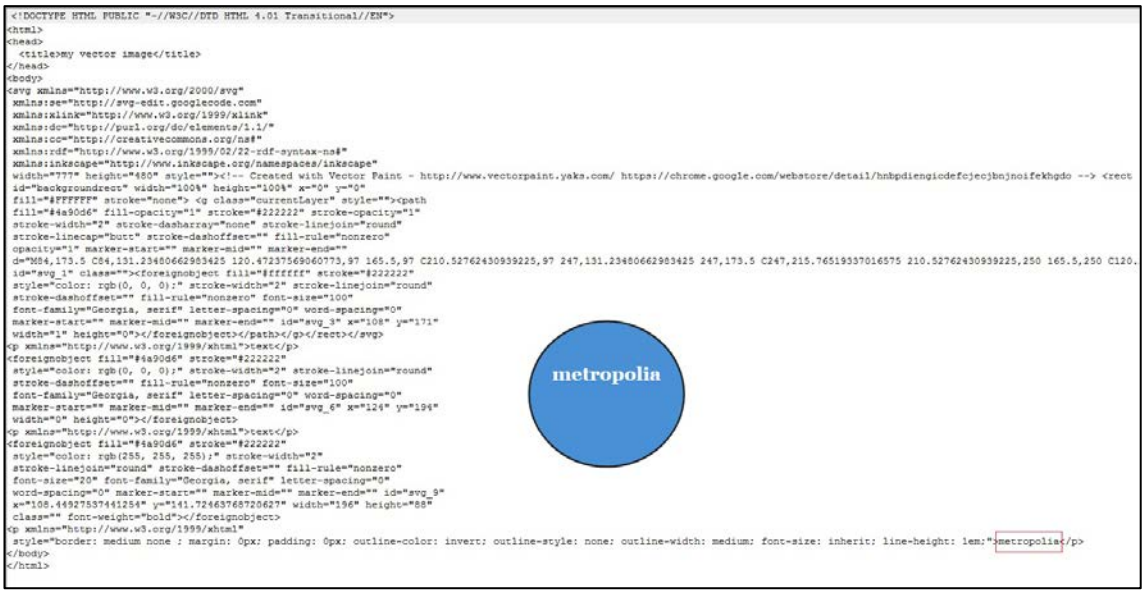


Figure 7. SVG image with blue circle and a Metropolia text

Despite its lightweight size and comprehensive browsing support, SVG was not perfect. Along with the JavaScript, the browsers used DOM (Document Object Model) as well. DOM is a programming interface which helps JavaScript to change web pages dynamically. Unfortunately, controlling the code with the DOM slows down the process when there are thousands of objects on the screen (Perna, 2016).

2.4.2 Canvas

Five years after the SVG images were introduced, Canvas technology came out to compete. Canvas technology came out with the HTML5 language and it was a fresh and dynamic style of drawing elements to the screen. Whereas the SVG figures were created with XML language, figures made with Canvas were written in JavaScript. The JavaScript code first defines the drawing area where the HTML page is drawing the Canvas. Then JavaScript commands are called to draw figures based on the X and Y coordinates. Canvas has ready-made commands for example to lines, circles, and text (Hawkes, 2011). Figure 8 shows how simple a structure of the Canvas code can be when drawing the blue circle.

```html
<body>
    <canvas id="myCanvas" width="178" height="200"></canvas>
    <script>
      var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');
      var centerX = canvas.width / 2;
      var centerY = canvas.height / 2;
      var radius = 70;

      context.beginPath();
      context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
      context.fillStyle = 'blue';
      context.fill();
      context.lineWidth = 0;
      context.strokeStyle = '#003300';
      context.stroke();
      context.font = '15pt Calibri';
      context.fillStyle = 'white';
      context.fillText('Metropolia', 43, 100);

    </script>
  </body>
```

Figure 8. Blue Metropolia circle made with HTML5 canvas technique

The size of the plot area is specified between the Canvas tags (see Figure 8). Between the script tags, the drawing of the circle itself is done via help of the coordinates and built-in commands.

Both SVG and Canvas have their own supporters in the world of web technologies and data visualization. In which circumstances it is smarter to use what, it depends on the scope of use. Canvas does not have the DOM object model to lower the performance. Learning curve to drawing is also low. SVG, on the other hand, is based on the vector graphics so that the patterns are easily modified. SVG has also a capability to change itself dynamically in the browser without unnecessary reloading (Hawkes, 2011).

2.4.3 Neo4j Graph Database

The third frequently mentioned term in the data visualization scene is Neo4j. It is the most popular graphical database from the Swedish company Neo4j, Inc. It differs from the traditional databases in such a way that the data is stored as nodes and relationships. Relationship always stands between two nodes. Nodes can have several relationships. Both the nodes and the relationships may have multiple properties (Bruggen, 2014),

No wonder if the structure sounds familiar. Many interactive data visualization interfaces have the same structure: Nodes and their relationships. Here are the Neo4j's three basic objects:

- Nodes (actual records)
- Relationships (edges connecting nodes)
- Properties (data values linked either nodes or relationships)

As shown in Figure 9, Neo4j has its own built-in data visualization interface. Metropolia and Vesa-Ville Välimäki circles represent the nodes in the picture. The STUDIES text means the relationship which Vesa-Ville has with the Metropolia UAS.

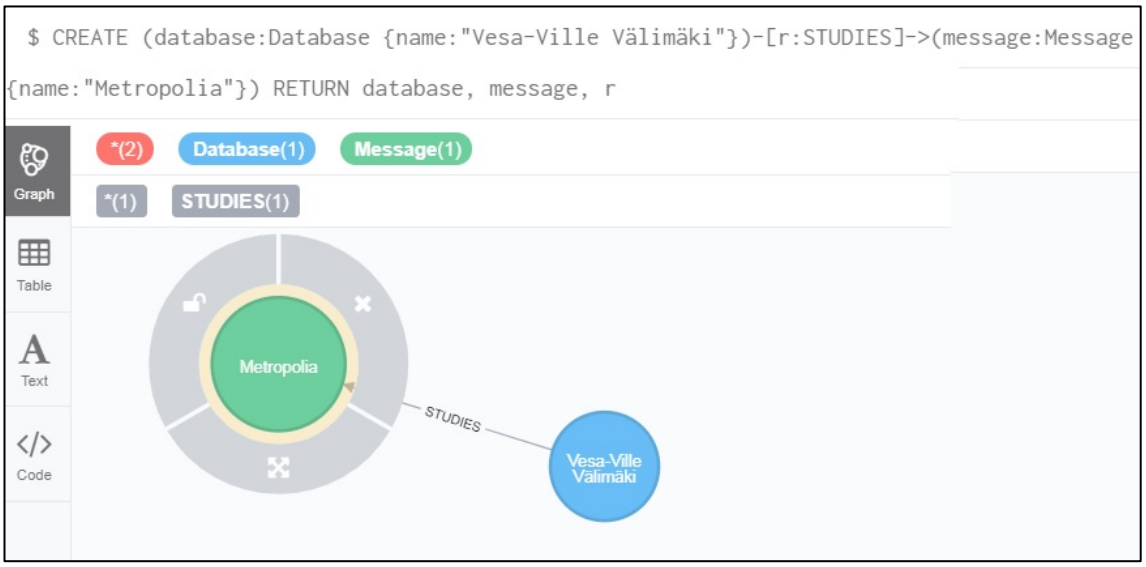


Figure 9. Graph database example made with Neo4j sandbox

Neo4j is used by many famous web sites such as eBay, AirBnb and Walmart. In addition, Neo4j offers its product in open source style under the APGL license. That means the companies that use Neo4j for free will have to open their code for everyone (Neo4j, 2017).

2.4.4 Big Data

Almost everything is leaving data behind today. Phones collect data, cars collect data, even sport performances collects data, especially if people are equipped with sensors. Living in a world where it is not a big issue if clothes and washing machines are collecting

data is near. All these data collections are forming the part of today's trendy word: Big Data (Marr, 2015).

As it was mentioned in Chapter 2.2, the need for analyzing Big Data has attracted data visualization developers come out from their garages. Nowadays prettier and faster user interfaces are seen than ever before. Visualization, however, is not the only thing that one should focus on when parsing the Big Data. For example, personal data is today full of interesting information like location, text, connections and behavior (Marr, 2015).

Data stored in the traditional database consists an information such as:

- First Name
- Last Name
- Address
- Phone Number

But today's databases with the Big Data can be filled with the information such as follows:

- Audio files
- Video files
- Images
- Documents

Back then, traditional databases were reserved for about 10 gigabytes of space. Today, one Boeing 737 airplane collects 240 terabytes of data for a flight. These figures still lose clearly to the giants of social media. For example, Facebook collects fresh data over 500 terabytes during a day (MongoDB, 2017).

2.4.5 Open Data

In addition to the Open Source code, the data visualization communities are also full of Open Data. Characteristics to Open Data is:

- Available to anyone
- Free to use
- Free to modify

- Distribute the data for own good

Typical Open Data distributors are cities, schools and public sector. They share data such as traffic information, geography locations and sociological details. Easily downloadable Open Data is also utilized by many private companies. They can re-use the data and benefit in the terms of money. It is not forbidden (Portal, 2015).

3 Evaluation of Modern JavaScript Libraries

There are many tools available for data visualizing. The tool used should be selected based on the needs of data usage. Here are the things that influences the choice of the data visualization application (Dormehl, 2014):

- The amount of data
- Format of the data
- What needs to be analyzed
- The data interconnections and interrelationships

With the help of this information, choosing four known JavaScript libraries to our tests were easy. These libraries have the best potential to meet the challenges:

- D3.js
- Sigma.js
- Linkurious.js
- Ogma

D3.js was an obvious choice for the first selection because it is one of the most popular JavaScript libraries in the world and capable to do interactive visualizations (Murray, 2013). Among the hundreds of other JavaScript libraries, the Bashooka.com website had a helpful list of 25 best JavaScript graphical libraries. Sigma.js was mentioned there and was chosen to this study because of its capabilities to do network exploring. Pages also mentioned the Linkurious.js which was chosen as the third option because of its massive collection of interactive examples. The developers of Linkurious has also made the Ogma library so that was selected as the fourth candidate. The choices from the Bashooka.com were supported also by the reviews and comments which were found from all the miscellaneous books, articles and online discussions during the years 2016-2017 (Wijaya, 2016).

3.1 Testing Equipment

For the tests, only one laptop and a web server were needed. In order to get realistic response times, using of local web server did not come to the question. Virtual server behind the Internet Service Provider was used. The test set-up was the following:

Client (Dell E7450) side specifications:

- OS: Windows 7, 64 bit
- CPU: 2,30 GHZ (Core i5-5300U)
- RAM Memory: 8 GB DDR3L
- GPU: HD Graphics 5500
- Resolution: 1920 x 1080 pixels

Server side specifications:

- OS: Centos 6 23 bit
- CPU: 2.4Ghz (1 core)
- RAM Memory: 1Gb
- Bandwidth: 100 GB
- Web Server: Apache 2.4

The connection used between the client and the server was 100/10 MBit Internet connection. That means that the download speed was 100 Mb per second and upload speed 10 Mbps. Response time was measured to be 6 milliseconds.

3.2 D3.js

In 2011, Heer and Bostock announced that they have developed a new visualization library with their colleagues. The open source library was named to D3.js. Its advantages compared to its predecessors were following (Murray, 2013):

- Easier to debug
- Faster and more agile in node handling
- More visualization options
- Possibility to use CSS styles (Cascading Style Sheets is a method how to decorate an HTML page)

D3.js was technically overwhelming what comes to the co-operation with the most advanced browsers. D3.js conquered the world, especially with the help of their active community. Good documentation and multiple examples (see Figure 10) encouraged users to make their own tunes and share them in Github. Currently D3.js is the Github's fourth most acquired JavaScript package and it has been downloaded 17,370 times (Skau, 2013).

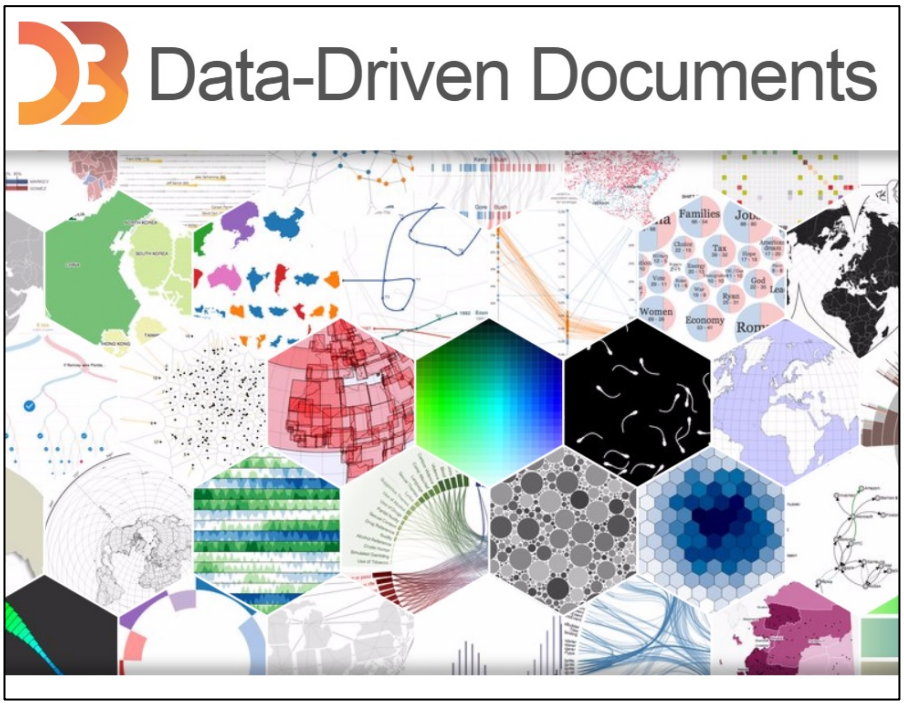


Figure 10. Collection of D3.js examples (Bostock, 2017)

Despite the success, D3.js was not perfect. The handling of objects based on the DOM interface proved to be painfully sluggish. Usability began to suffer when the number of elements on the screen began to rise around 10,000 pieces. Part of the slowdown was also caused by the SVG images, the weaknesses of which were mentioned Chapter 2.4.1. (Skau, 2013) SVG images were also poorly supported by most browsers at that time (Smith, 2015).

The popularity of D3.js has also caused some disadvantages. Finding good data visualization examples from their collections was challenging. D3.js's home page contains over 500 examples but it was hard to see from the thumbnails what they really had to offer. Link to examples were redirected to Github or to manufacturer's home page, which made the search even more difficult (D3.js, 2017).

However, one of the examples got the attention. The example was called "Interactive force layout" and can be seen in Figure 11. A closer look shows it was the D3 Process Map example edited by James Nylen. It told the story about the well-known Les Miserables novel and its personal connections. Although there was not a massive amount of data on the screen, there was an excellent style of fading the background information.

This feature is missing from surprisingly many competitors, even if it is very easy way to make the navigation clear (Nylen, 2016).

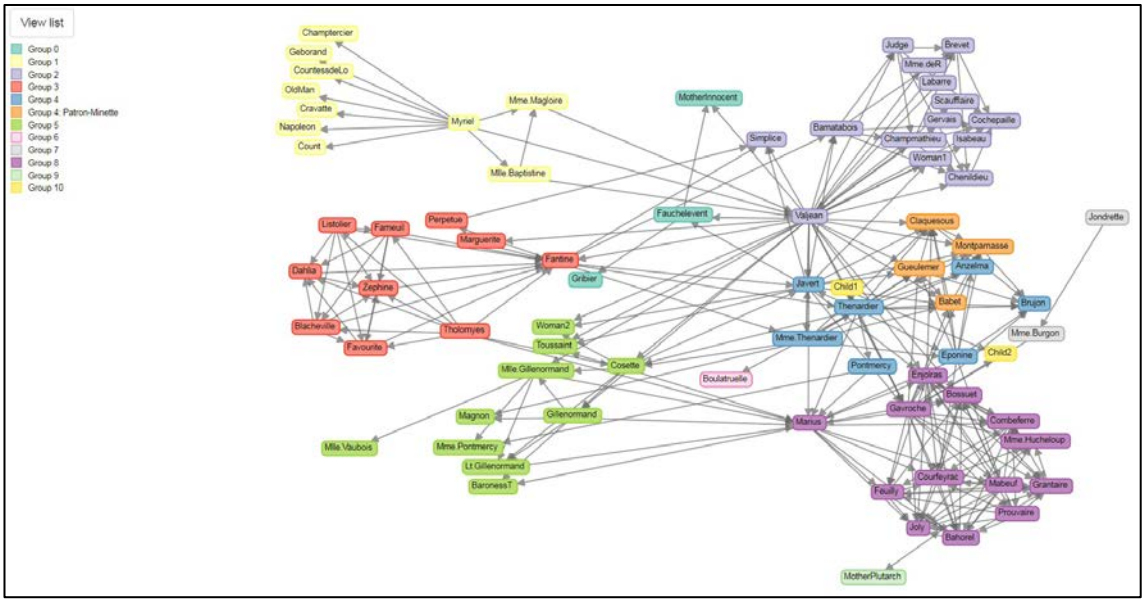


Figure 11. D3 Process Map and their connections (Nylen, 2016)

Process Map package were downloaded to test engine and its contents were investigated. First, it was noticed that there was a lot of PHP (Hypertext Preprocessor) files in it. Although the PHP is a popular programming language for a web development, it required some computing skills to get the example to work on a web server.

After the successful launch of the PHP pages, the example worked quite painlessly. However, the challenges began when it was time to start to edit the example. As shown in Figure 12, the Process Map main page had numerous references to different JavaScript and CSS files.

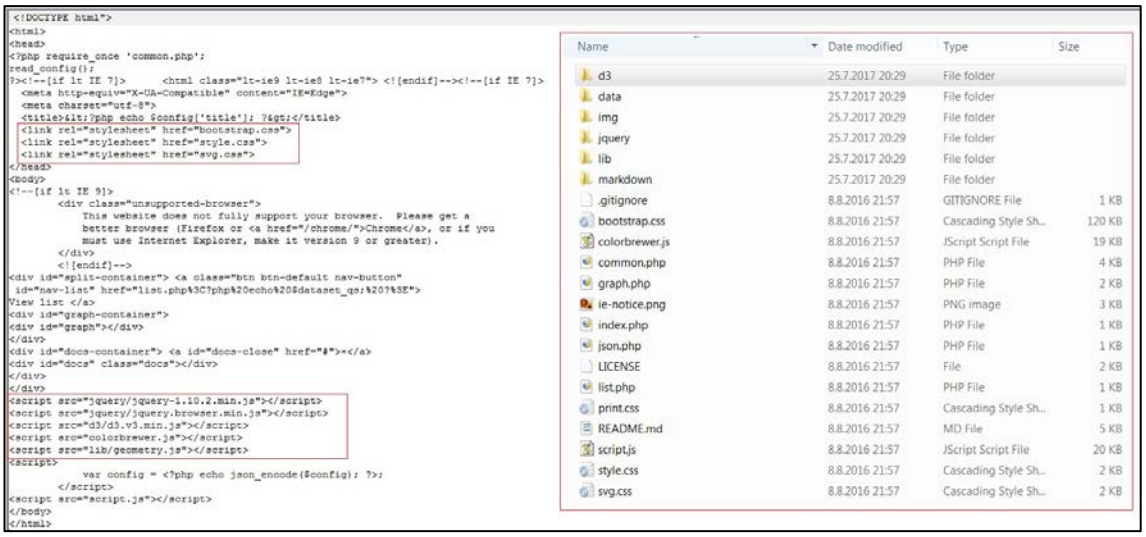


Figure 12. Code and the tree view of the Process Map example

Because of the increasing programming challenges, further investigations of D3.js was decided to stop. One of the criteria for the thesis was to find a JavaScript library that would be easy to edit, and despite the strong popularity of the library, the editing of D3.js examples are not at the same level of ease compared to competitors.

3.3 Sigma.js

Jeff Heer and Michael Bostock have been involved in a number of success stories related to data visualization. There are still other skillful experts in the field. One of them is Alexis Jacomy who made his reputation inside the Gephi community. Gephi was an open-source data visualization software that was popular among the academic scientists (Bastian, 2016). In 2012, Jacomy published a JavaScript library which was focusing on the visual side of the data visualization as the beautiful Figure 13 shows. The library was called Sigma.js. (Labs, 2012). Figure 13 shows an image from the Sigma.js home page.

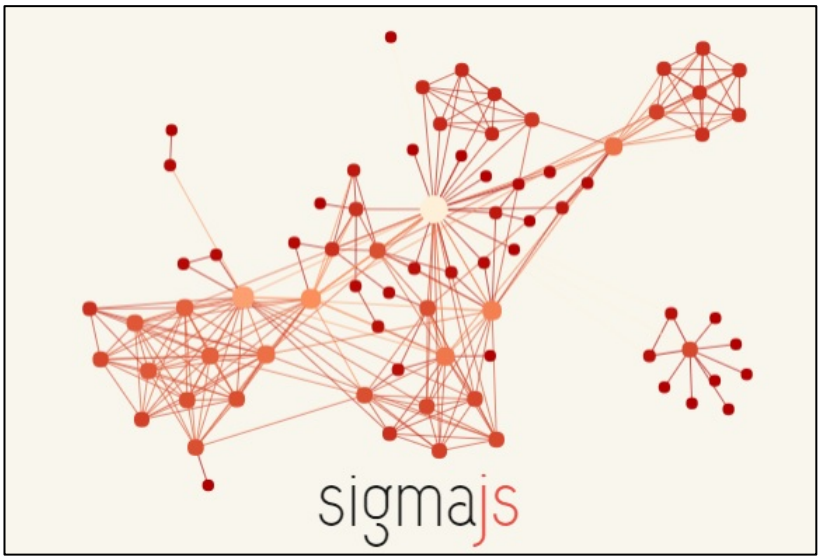
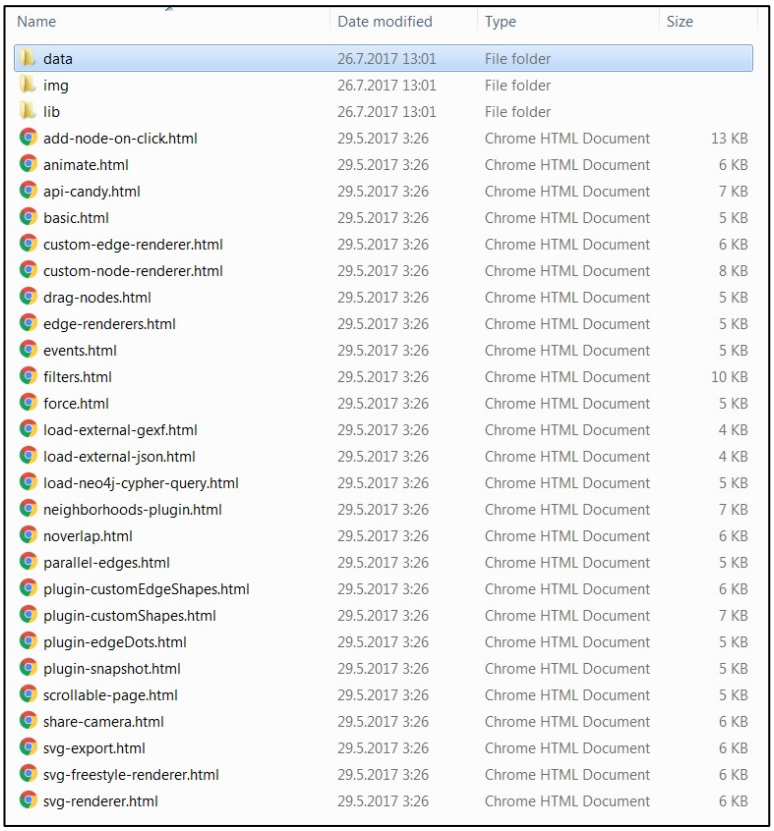


Figure 13. Image from the Sigma.js home page (Jacomy, 2012)

Sigma.js was welcomed very well in the data visualization community. People were fascinated by its features, e.g. (Marques, 2014):

- Very fast and agile to render large figures
- Support for Canvas and webGL
- Advanced interactive features with a mouse like
    - Clicks and double-clicks
    - Mouseover effects
    - Zoom
- Visually beautiful examples

Sigma.js was decided to be taken to the tests. Unfortunately, there were no any images available from the available templates in their site. Besides, the community of Sigma.js was very small. In the Github the library has been downloaded only 1 162 times. Nevertheless, package was downloaded and all the 26 examples were extracted. The decision was to test all the examples shown in Figure 14 manually.



Name	Date modified	Type	Size
data	26.7.2017 13:01	File folder	
img	26.7.2017 13:01	File folder	
lib	26.7.2017 13:01	File folder	
add-node-on-click.html	29.5.2017 3:26	Chrome HTML Document	13 KB
animate.html	29.5.2017 3:26	Chrome HTML Document	6 KB
api-candy.html	29.5.2017 3:26	Chrome HTML Document	7 KB
basic.html	29.5.2017 3:26	Chrome HTML Document	5 KB
custom-edge-renderer.html	29.5.2017 3:26	Chrome HTML Document	6 KB
custom-node-renderer.html	29.5.2017 3:26	Chrome HTML Document	8 KB
drag-nodes.html	29.5.2017 3:26	Chrome HTML Document	5 KB
edge-renderers.html	29.5.2017 3:26	Chrome HTML Document	5 KB
events.html	29.5.2017 3:26	Chrome HTML Document	5 KB
filters.html	29.5.2017 3:26	Chrome HTML Document	10 KB
force.html	29.5.2017 3:26	Chrome HTML Document	5 KB
load-external-gexf.html	29.5.2017 3:26	Chrome HTML Document	4 KB
load-external-json.html	29.5.2017 3:26	Chrome HTML Document	4 KB
load-neo4j-cypher-query.html	29.5.2017 3:26	Chrome HTML Document	5 KB
neighborhoods-plugin.html	29.5.2017 3:26	Chrome HTML Document	7 KB
noverlap.html	29.5.2017 3:26	Chrome HTML Document	6 KB
parallel-edges.html	29.5.2017 3:26	Chrome HTML Document	5 KB
plugin-customEdgeShapes.html	29.5.2017 3:26	Chrome HTML Document	6 KB
plugin-customShapes.html	29.5.2017 3:26	Chrome HTML Document	7 KB
plugin-edgeDots.html	29.5.2017 3:26	Chrome HTML Document	5 KB
plugin-snapshot.html	29.5.2017 3:26	Chrome HTML Document	5 KB
scrollable-page.html	29.5.2017 3:26	Chrome HTML Document	5 KB
share-camera.html	29.5.2017 3:26	Chrome HTML Document	6 KB
svg-export.html	29.5.2017 3:26	Chrome HTML Document	6 KB
svg-freestyle-renderer.html	29.5.2017 3:26	Chrome HTML Document	6 KB
svg-renderer.html	29.5.2017 3:26	Chrome HTML Document	6 KB

Figure 14. Sigma.js examples from the version package 1.2.0.

Relatively many of the examples showed the speed and agility of the graphics engine. Their usefulness and the potential for further development were quite bad. However, one of the examples seemed to fit for the purposes. The example was called Drag nodes and it was convincing, with the capability of drag nodes everywhere on the screen. Along with that it looked visually very aerial and usable as noticed in Figure 15.

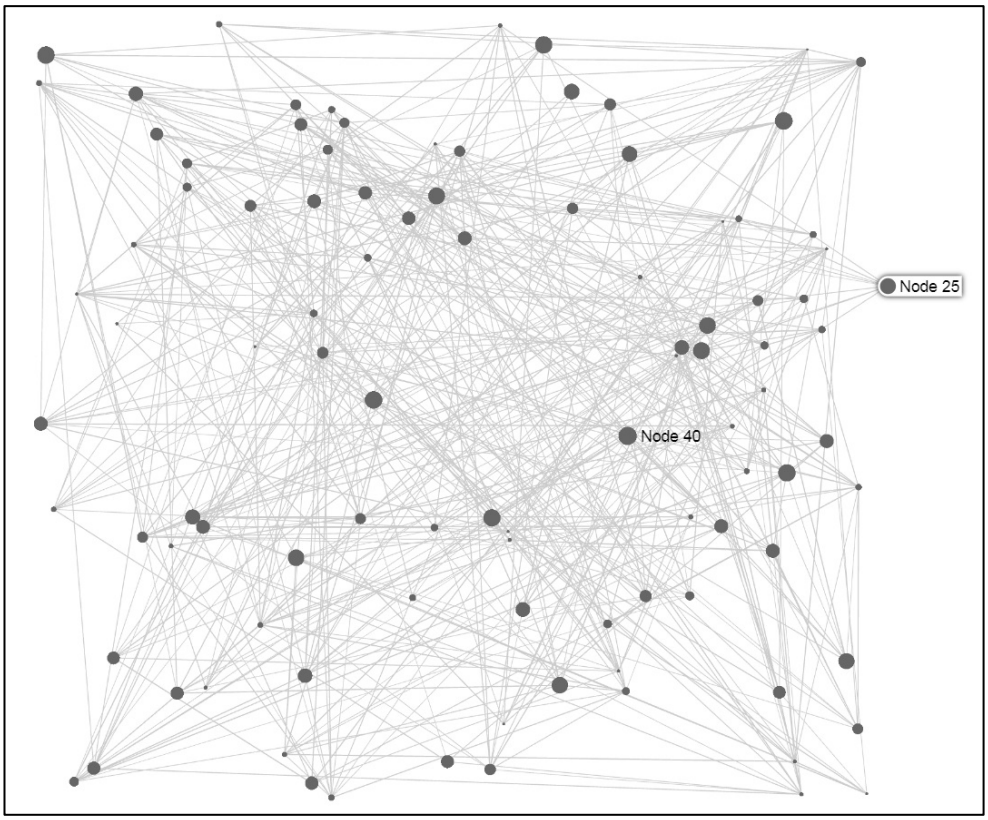


Figure 15. Sigma.js and drag-nodes.html example

After a quick test, dragging the nodes in the visualization worked fluently. Because there was no noticeable slowdown, the decision to increase the load was clear. As shown in Figure 16, the number of nodes (N) was only 100 by default and the number of edges (E) was 500. Test was started again after adding the number of nodes to 1000.

```
<script>
/**
 * This example shows how to use the dragNodes plugin.
 */
var i,
    s,
    N = 100,
    E = 500,
    g = {
      nodes: [],
      edges: []
    };
```

Figure 16. The amount of nodes and edges in the drag-nodes example

1000 nodes did not affect to usability so the number of nodes was increased to 10 000. The user interface then began to slow and the problems of zooming were getting worse.

Some of the dragging nodes started to disappear from the display and the slowness almost doubled. The Sigma example with 10 000 nodes is shown in Figure 17.

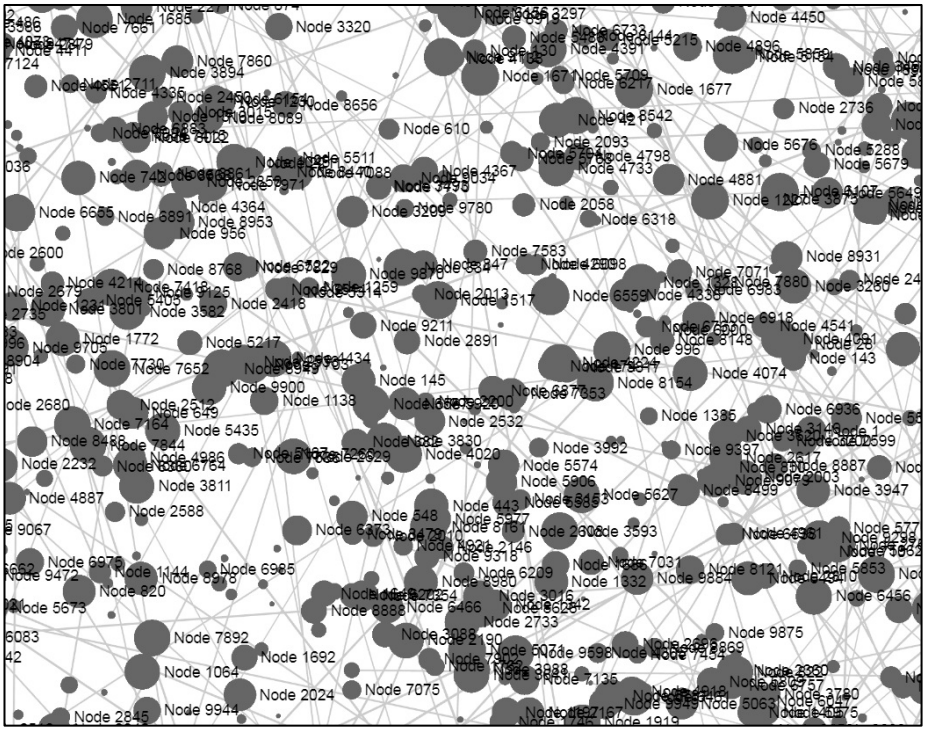


Figure 17. Sigma drag nodes example and 10 000 nodes

As shown in Figure 17, the user interface was too confusing and the tests with Sigma.js was decided to stop.

3.4 Linkurious.js

The Sigma.js JavaScript library had a couple of fans in the community, including French gentlemen Sébastien Heymann, David Rapin and Jean Villedieu. Together, they decided to set up a startup company called Linkurious SAS. The purpose of the company was to provide visual tools for data analysing. They started building their own products based on Sigma.js (Linkurious.js, 2017).

Linkurious SAS has a flagship product called Linkurious Enterprise. This pre-packaged solution promises to take care of the customer needs with the following features (Linkurio.us, 2017):

- Ready-made, clear and fast graphical user interfaces

- Possibility to customize the graphical side
- Monitoring of suspicious connections and automatic alerts
- Graphical user interface for editing data source
- Comprehensive filtering options
- Graphic analysis tools for identifying connections

In addition to Linkurious Enterprise, they maintain an open code community around the Linkurious.js library. Not surprisingly, the Linkurious.js library is also the main engine for Linkurious Enterprise (Home, 2017). Linkurious.js is based on Sigma.js added with extra filters and plugins (Linkurious.js, 2017).

Linkurious Enterprise was not available without registration but downloading the Linkurious.js library with examples was done. Going through all the examples, the most appropriate model was definitely the example called "Drag nodes". The model is based on the same Sigma.js example as the one discussed in Chapter 3.3 (SAS, 2017)

Opening the Drag nodes code revealed that example uses 19 different Linkurious plugins along the Sigma.js library. The code and the plugins are shown in Figure 18.

```
<body>
<!-- START SIGMA IMPORTS -->
<script src="../src/sigma.js"></script>
<!-- END SIGMA IMPORTS -->

<!-- START LINKURIOUS PLUGINS -->
<script src="../plugins/sigma.plugins.dragNodes/settings.js"></script>
<script src="../plugins/sigma.plugins.dragNodes/sigma.plugins.dragNodes.js"></script>
<script src="../plugins/sigma.helpers.graph/sigma.helpers.graph.js"></script>
<script src="../plugins/sigma.plugins.activeState/sigma.plugins.activeState.js"></script>
<script src="../plugins/sigma.plugins.select/sigma.plugins.select.js"></script>
<script src="../plugins/sigma.plugins.keyboard/sigma.plugins.keyboard.js"></script>
<script src="../plugins/sigma.renderers.linkurious/settings.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.labels.def.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.hovers.def.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.nodes.def.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.nodes.cross.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.nodes.diamond.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.nodes.equilateral.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.nodes.square.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.nodes.star.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.edges.def.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.edges.curve.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.edges.arrow.js"></script>
<script src="../plugins/sigma.renderers.linkurious/canvas/sigma.canvas.edges.curvedArrow.js"></script>
<!-- END LINKURIOUS PLUGINS -->


<div id="container">
  <style>
    #graph-container {
      top: 0;
      bottom: 0;
```

Figure 18. Inside the code: Calling of Sigma library and Linkurious plugins

After loading the plugins, location of the visualization in the web site is defined inside the div and style tags. The div tag name is set to "container" which is called later on the site to perform the desired JavaScript code. Style tags determine how far from the browser borders the visualization is located. As shown in Figure 19, in this case all four main directions have received the value zero. In practice, the application will fill the entire screen and data visualization will get the full capacity for use.

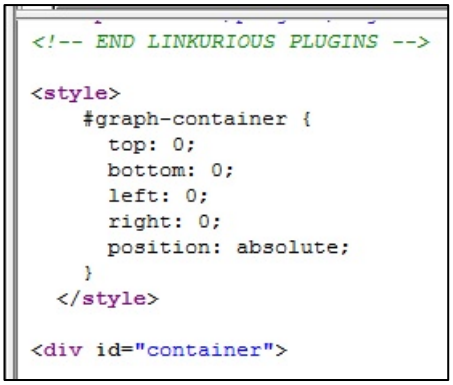


Figure 19. Linkurious.js and defining the location of the visualization

The next interesting thing was what kind of default settings Drag nodes had with an example pattern. From the comments of code in Figure 20 it was already possible to guess the result. The application drew the figure randomly every time the page was refreshed. In Chapter 3.3 it was already shown that by increasing the variable N more nodes were obtained. That is how putting more and more load to Linkurious.js engine can be done.

```
<script>
/**
 * This example shows how to use the dragNodes plugin.
 */
var i,
    s,
    N = 100,
    E = 500,
    g = {
      nodes: [],
      edges: []
    };

// Generate a random graph:
for (i = 0; i < N; i++)
  g.nodes.push({
    id: 'n' + i,
    label: 'Node ' + i,
    x: Math.random(),
    y: Math.random(),
    size: Math.random(),
    color: '#666'
  });

for (i = 0; i < E; i++)
  g.edges.push({
    id: 'e' + i,
    source: 'n' + (Math.random() * N | 0),
    target: 'n' + (Math.random() * N | 0),
```

Figure 20. Linkurious.js example and number of nodes in the screen

The last easily modified property in the code was settings. By editing the settings, the color of the drawing, the thickness of the edges and the mouse-over effects of the nodes were influenced. The settings are shown in Figure 21.

```
sigma.renderers.def = sigma.renderers.canvas;

s = new sigma({
  graph: g,
  renderer: {
    container: document.getElementById('graph-container'),
    type: 'canvas'
  },
  settings: {
    dragNodeStickiness: 0.01,
    borderSize: 2,
    outerBorderSize: 3,
    defaultNodeBorderColor: '#fff',
    defaultNodeOuterBorderColor: 'rgb(236, 81, 72)',
    enableEdgeHovering: false,
    edgeHoverHighlightNodes: 'circle',
  }
});
```

Figure 21. Linkurious.js and settings of the graph

Drag Nodes example with 10 000 nodes were tested. Elements appeared on the screen at less than 0,5 seconds which was really fast. The screen looked the same as in the example shown in Figure 17, but the navigation was much faster than in Sigma.js. As a result, Linkurious.js was taken to the final testing which is described in Chapter 5.

3.4.1 Case Nasa

In 2015, Linkurious got a big and interesting client: NASA. Inside the NASA organization there were lot of different projects where massive amounts of information were available. The learned lessons, errors, and good practices were collected in a shared database called "NASA Lessons Learned System". It was a good idea in theory but in practice it was not used enough (Carey, 2017).

The problem was not that their search engine was not fast enough or accurate. It was not just agile enough in order to retrieve information via multiple search terms. Each term had to apply separately and after that it searched from more than 20 million documents. Every document had to select and save it for later research. This practice was frustrating and took lot of time from the employees (Carey, 2017).

NASA decided to renew the way data is retrieved from their database. The search logic was changed and the data was converted to a Neo4j database. As it was described in Chapter 2.4.3, it was possible to record the different links between the documents and their relationships in Neo4j. Now there was only a need for a tool that could easily search and display the search results on the screen (Carey, 2017). NASA acquired the ready-made solution of Linkurious as Figure 22 shows.

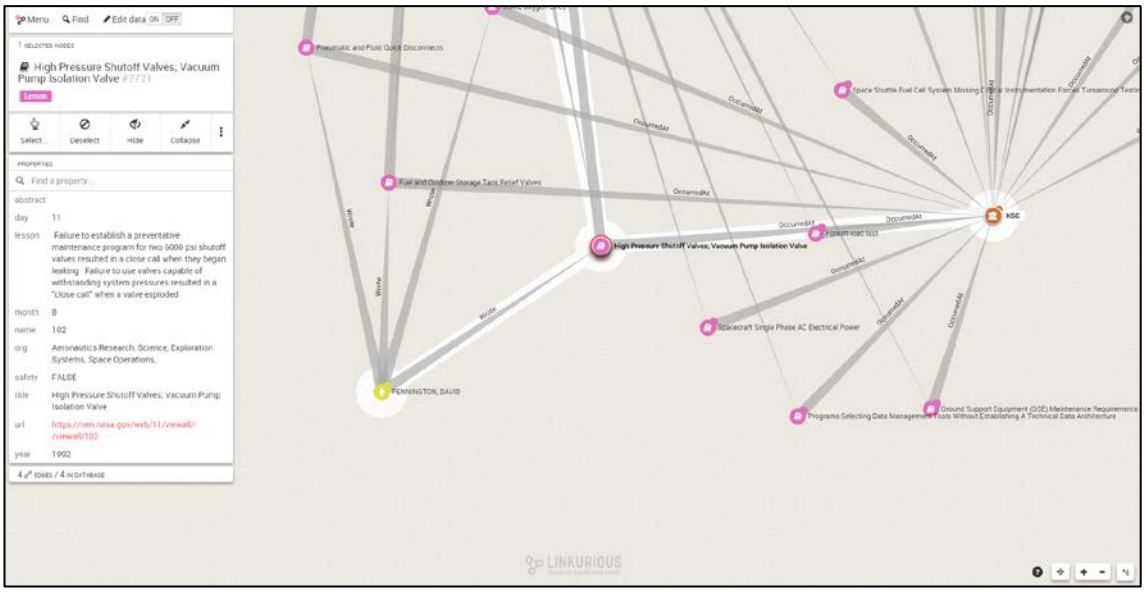


Figure 22. Multiple search terms in NASA with Linkurious (Villedieu, 2015)

Since the acquisition, employees have had a more effective way of searching for information. This saves working time and researchers have the opportunity to do more valuable things in their future work (Carey, 2017).

3.4.2 Panama Papers Investigation

The most famous event in the world of Interactive Data Visualization was undoubtedly the "Panama Papers". There were 11.5 million documents leaked from the Panamanian law firm. The data was about the companies and their funds invested in tax havens. There were 370 journalists in 76 countries who wanted to participate on the data exploring sessions. In this case, good tools played also an important role (Shane, 2016).

The Linkurious Company was invited to join the Panama investigation. The invitation did not come as a surprise because they were already partners with the international research journalist association ICIJ. They had also experience about HSBC bank data loss case in 2007. The amount of data in the Panama Papers was about 2.6 terabytes but Linkurious did not fright. They did have experience of Big Data, thanks to that NASA case which was mentioned in Chapter 3.4.1. (Panama Papers: How the French start-up Linkurious helped to make it all happen, 2016)

Panama Paper investigation included data as much as:

- Over 4 million emails
- 3 million database files
- 2 million PDF-files
- Over 1 million image files
- Over 300 000 text files

Fortunately Linkurious did not have to do everything by themselves. ICIJ translated all of the above data into the machine readable format. Neo4j, which was presented in Chapter 2.4.3, was offering their graphical database as a data store. ICIJ wanted to distribute the material to reporters so that analyzing would be as easy and safe as possible. To be able to do all of that, Linkurious Enterprise was the best option. In addition to research and analysis, the Linkurious Enterprise software helped journalists also create infographic images in their publications. These same pictures, as in Figure 23, were also exchanged between the journalists. All the researchers participated in the global collaboration (Panama Papers: How the French start-up Linkurious helped to make it all happen, 2016).

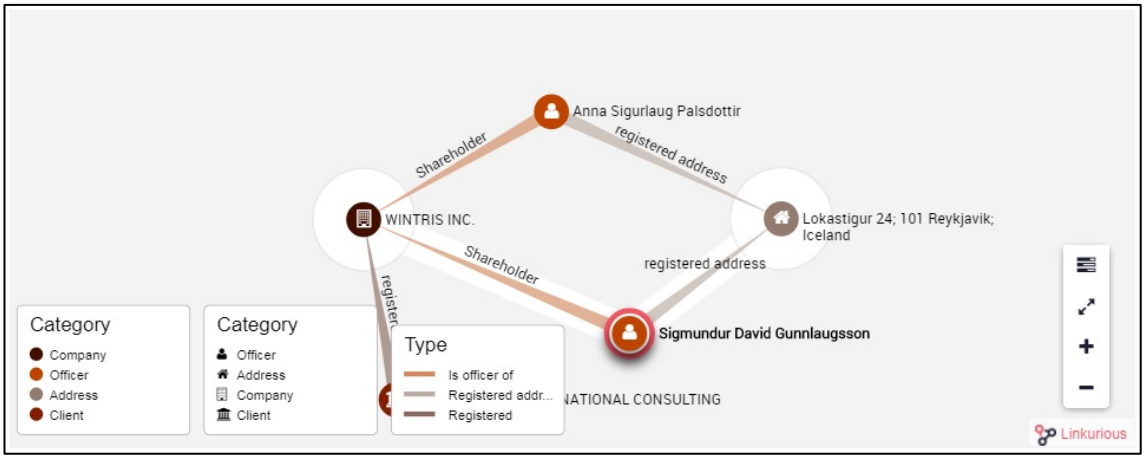


Figure 23. Exploring the secrets of the Prime Minister of Iceland (Heymann, 2016)

An interesting detail in the investigations was the fact that Linkurious employees did not see any of the leaked data. All data analysis and traffic went through the encrypted channels. Linkurious Enterprise also took care of user management. None of the outsiders had access to the material (Panama Papers: How the French start-up Linkurious helped to make it all happen, 2016).

3.5 Ogma

Heymann, the founding member of Linkurious, published an announcement on their company's blog site in October 2016. They had published a new JavaScript library called Ogma. It was dedicated specifically for the interactive research of the large material. In the same announcement he summed up the problem which was also the core of this thesis (Heymann, 2017):

> State of the art visualization libraries for the Web are unable to display graphs with more than 10 000 nodes and edges. With Ogma is it now possible to display more than 100 000 nodes and 100 000 edges. (Heymann, 2017).

This publication came at the perfect time from the thesis point of view, and Ogma was included in the research. It provided the necessary belief that things will go better way in the world of the data visualization engines, like the fresh looking interface in Figure 24 shows.

Figure 24. Ogma Demo with Eurovision 2015 votes example

What is under the hood of Ogma was obviously an interesting question. What are the aspects that promised better results than their predecessors? Ogma's documentation was investigated but no technical details were available. It was only told that the code is based on the Linkurious.js library. However, logically meaningful changes that make the

performance raise were reported openly. The logical changes are handled in Chapter 3.5.1 (Heymann, 2017).

3.5.1 Enhancements of Ogma

Instead of bringing all of the massive data to the screen at once, Ogma's approach was slightly different. They have built their JavaScript library so that unnecessary things will automatically stay out of the screen. This would speed up the visualization performance and navigation would be more agile. Developers would then have one thing less to worry about (Heymann, 2017).

The library have been designed to take real-time data variations into account. So if the data source changes, the Ogma library can show the changes live at the screen. Practically this means that it could show and hide nodes automatically. The developers have been taken into account better because the Ogma’s code is now packaged into modules. A library made up of modules gives developers a better opportunity to make changes easier (Heymann, 2017).

3.5.2 Browsing the Ogma Templates

Ogma's homepage had a comprehensive range of pre-made examples. From the hundreds of examples, Neighborhood highlight example was chosen to the tests of this Thesis. As can be seen in Figure 25, there were lot of those promising features available which makes the Interactive Data Visualization great: Elegant appearance, fast-responsing functions and user friendly animations. Ogma has also been tested to be compatible with over 80 browser versions (SAS, 2017).

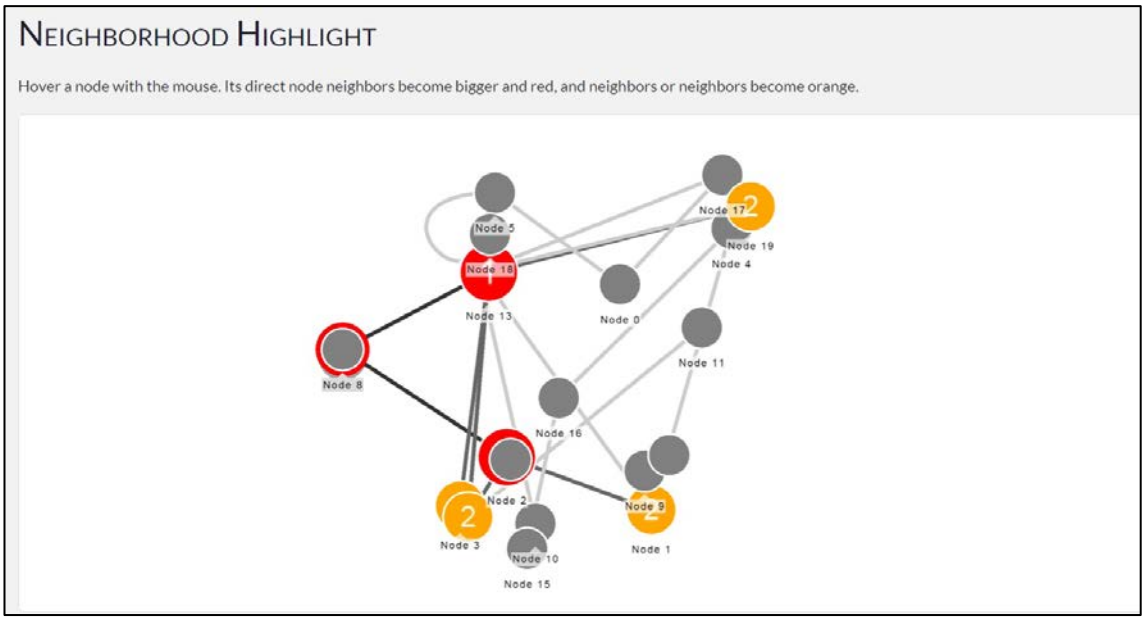


Figure 25. Neighborhood Highlight demo from Ogma examples website

The code producing the visualization was showed under every example in the Ogma's homepage. A quick review showed that manual editing was possible, so Ogma was decided to be taken to the final test introduced in Chapter 5.

4 Big Data selection

After the data visualization libraries were selected for final testing, the choice of the Big Data was started.

4.1 Selection of Data

For the test, as large a file as possible was needed. That was the efficient way to test the limits of visualization engines. In addition to the big size, the data had to be free and openly available. Although different types of files could be converted to another format, JSON file format was the one to look. JSON is the format that most of the JavaScript data visualization engines are supporting. Linkurious.js and Ogma made no exceptions.

A quick review showed that the Internet is not full of large and free data files. Many countries and cities have published open data interfaces, but most of them let users retrieve data only a small piece at a time. It would have been possible to combine multiple searches, but it would have taken too much time with the tools available.

4.1.1 San Francisco Data

One site turned out to be a relief for the researcher. The City and County of San Francisco offered an official and open data portal called SF OpenData. San Francisco's decision-makers believe, as in Figure 26, that publishing open data will help users to build better services and create new businesses. The use of data is allowed under the ODC Public Domain Dedication & License PDDL. It means that the data can be used freely for any purpose (Francisco, 2017).

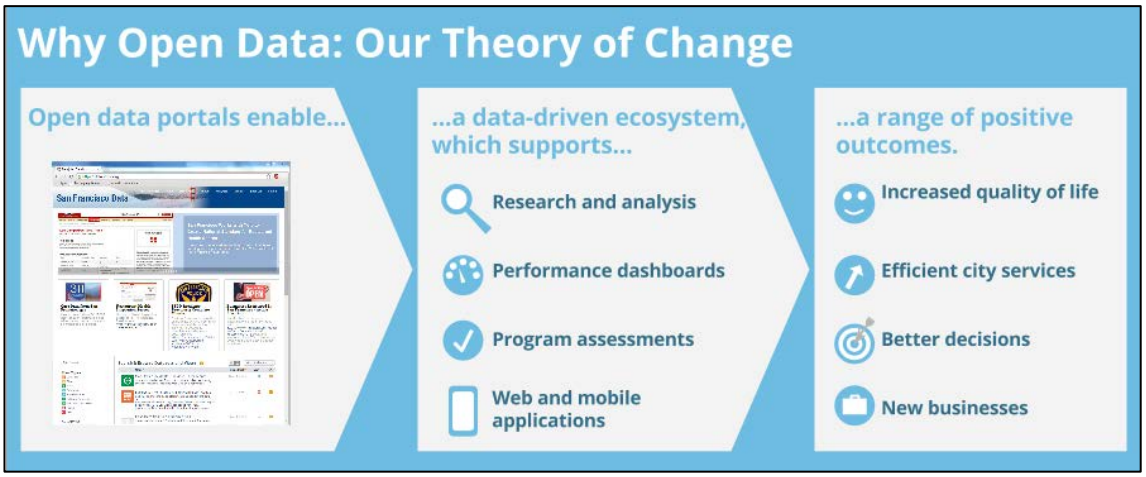


Figure 26. SF Open Data's Theory of Change (Francisco, 2017)

There were lots of different data files available in the SF Open data service. For example files dealing with the filming locations and emergency calls in San Francisco area. One particular file was however perfect for the present study: A file over 189 megabytes full of San Francisco City Lots. It contained detailed information about the City area lots such as street addresses and block ID numbers. The data can be viewed visually in the map shown in Figure 27. The black rectangles represents the boundaries of real estates.

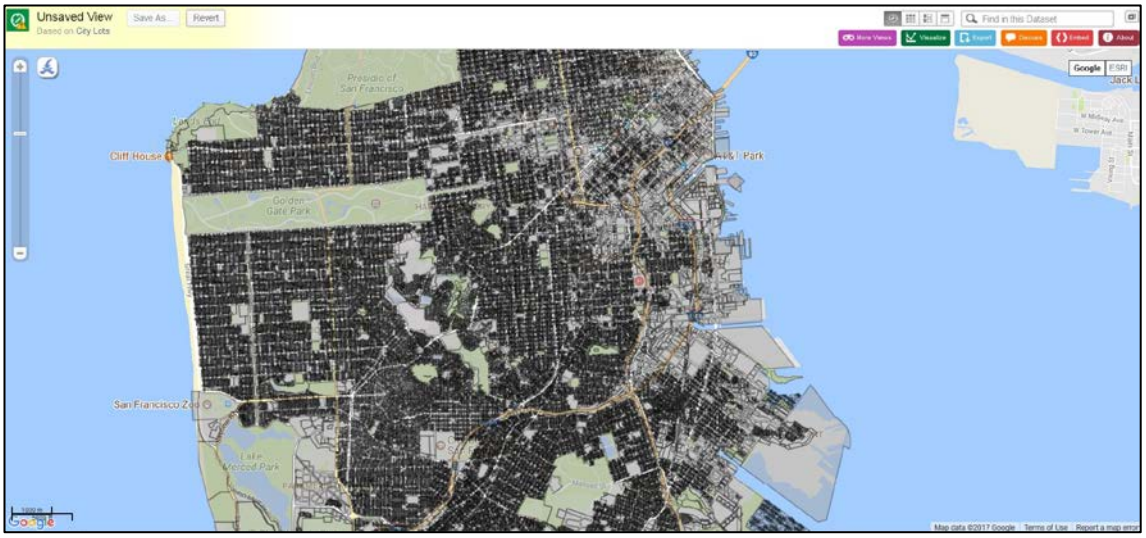


Figure 27. San Francisco Geographic Locations and Boundaries: City Lots

There were a number of download options available for the file. JSON format was chosen because it was the best option to visualization libraries. The test computer was facing

hard times with the Chrome browser (version 59.0.3071.115) when opening the file. Rendering the JSON file to the browser took 2 minutes and 24 seconds as shown in Figure 28.

Figure 28. Chrome Browser and 189 MB CityLots.json file

With the Windows Notepad application, the file opened just under one minute. The slow opening with the Chrome showed that the browsers play a big role when loading visualizations to the screen.

4.1.2 Modifying the JSON

A separate JSON editor was needed to facilitate the research of the JSON file. A free and lightweight editor was searched because that would not need to be installed on a test machine. A so-called portable executable version that works as a stand-alone without installed DLL files was under search. As shown in Figure 29, a program called JSONEdit was chosen. It contained the necessary visual elements needed: The text and the tree view.

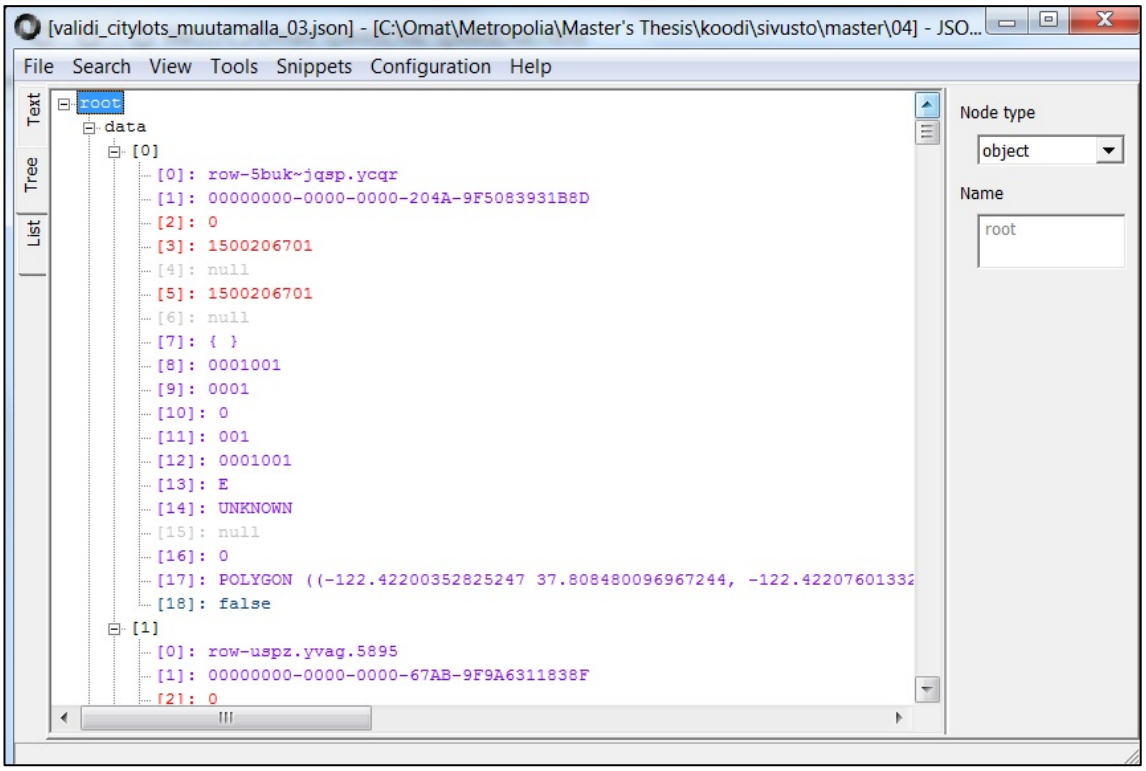


Figure 29. JSONEdit and tree view with the opened CityLots.json

JSONEdit opened the CityLots.json file to the text view in 15 seconds. The tree view could not opened it because the program indicated that there was not enough memory available. Fortunately that did not have any effect to tests. Only a small, 1000 rows sample file for the beginning was enough.

5 Testing Data Visualization Tool with Big Data

On the basis of information provided in Chapter 3, the decision about the libraries was made. They would be on the final test with the chosen Big Data. The main criterion for the library was how well the candidates handle a large number of nodes simultaneously on the screen. Despite the high popularity, D3.js did not meet the criteria because of the problems with the large data masses. Again, Sigma.js did not provide an example that would have been easy to modify to a user friendly interface.

Linkurious SAS was undeniably the company that fulfilled the demands best. Their fast JavaScript libraries and easily customizable sample applications were best suited to our goals. The most important criterion was the handling of big masses, and Linkurious gave some promising results in the preliminary study. The tests were decided to be done with both Linkurious.js and Ogma.min.js libraries.

5.1 Linkurious.js

The example of Linkurious.js "Drag nodes", mentioned in Chapter 3.4, proved to be a promising sample for further research. The entire Linkurious.js package (version 1.3.0.) was downloaded from the Github and was placed on a remote web server. Code worked right away so the next step was to go and see how easily the Drag nodes example was customizable.

5.1.1 Modifying Linkurious

After the look at the default settings of Drag nodes, it was time to edit the code so that it would allow a local JSON file to be used. Figure 30 shows how the objects were used to retrieve the content of the CityLots.json to the same web server. It was possible for the Chrome browser to read and to show information from a JSON file thanks to XHR object. Then the content was read to an array called myArr. Code snippets were placed inside the JavaScript tags just before the number of nodes selection started in the code.

```
<script>
/**
 * This example shows how to use the dragNodes plugin.
 */

var xmlhttp = new XMLHttpRequest();
var url = "CityLots.json";

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var myArr = JSON.parse(xmlhttp.responseText);
        myFunction(myArr);
    }
};
xmlhttp.open("GET", url, true);
xmlhttp.send();

var i,
    s,
    N = 100,
    E = 500,
    g = {
```

Figure 30. Placing the CityLots.json to an array

When the file was read to an array, it was time to count the rows. The variable “j” was assigned to represent the number of rows from the “data” field when looped through the JSON. Each rows included the information for one city lot. The variable j was finally assigned to the variable N because the latter shows the number of nodes in the default code. The code is viewable in Figure 31.

```
function myFunction(arr) {

    var j;
    for (j in arr.data) {
        //Number of json rows
    }

}

var i,
    s,
    N = j,
    E = 500,
    g = {
      nodes: [],
      edges: []
    };
```

Figure 31. Counting the rows from JSON file to variable N

It was time to launch the code with the first test material. It included 1000 rows of city lot information. Executing the page with Chrome for the first time took 2.81 seconds. The visualization worked very fluently and can be seen on the left side of the of Figure 32:

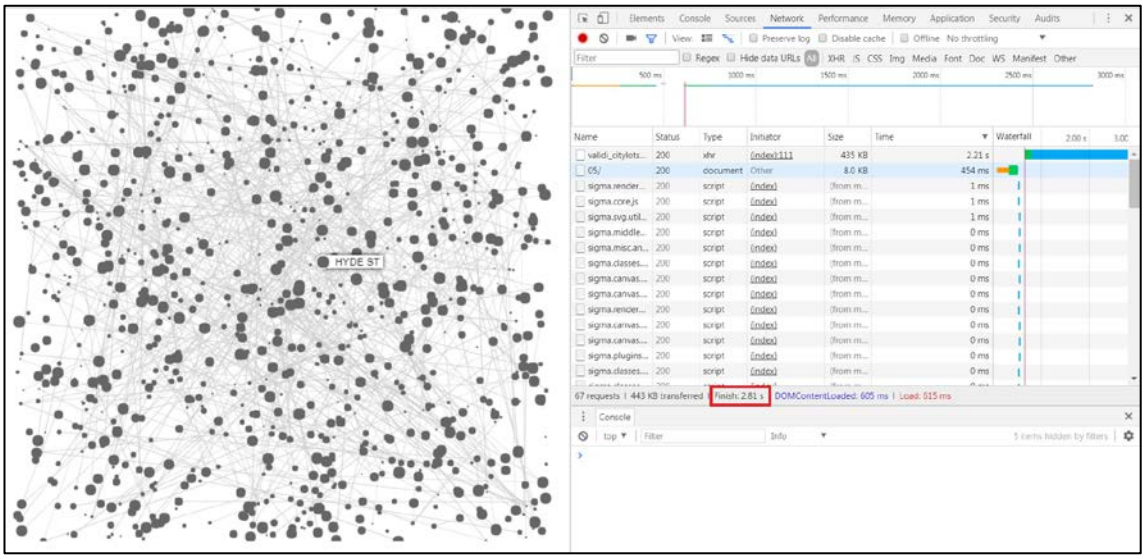


Figure 32. First Linkurious test with 1000 citylots rows

The following executions with the help of the browser's cache came into the finish in about 0.6 seconds. The conclusion was that the material of thousands City Lots worked with Linku-rious.js library. In the following tests, the number of rows was systematically increased. The results from the further tests are visible on Table 1:

Table 1. Measurement results of Linkurious.js.

The amount of City Lots	**Time (seconds)**	**User interface**
1 000	2,81	Working very smoothly
5 000	6,87	Working steadily
10 000	11,88	Slow but usable
25 000	25,45	Very stiff
50 000	51,13	Hardly usable
100 000	96	Useless

The results showed that the limit of Linkurious.js goes around 10,000 nodes. With 25 000 nodes on the screen, navigation was no longer pleasant. With 50 000 nodes the user interface was almost impossible. With Firefox browser (version 52.2.1) the response time was almost at the same level as Chrome, but usability was little bit behind. With

Internet Explorer 11, the results were worse than their competitors. For example, the data of 10 000 nodes took 15 seconds and was really stiff to use.

5.2 Ogma

The last candidate in review was Ogma, the latest addition to data visualization scene from Linkurious. The installation to the test machine was simple: HTML example page and a 437 KB JavaScript file called ogma.min.js was downloaded first. Then the files were placed to the same folder. The JavaScript file contained all the required functions from the Ogma library 1.5.3.

5.2.1 Modifying Ogma

During the code structure investigation of a Demo neighborhood example, it was pretty obvious that the engine was derived from the Linkurious.js library. The code shown in Figure 33 shows how easy it was to get familiar with the code. Some of the variables had even retained the same naming standard. The nodes were named as N and the edges were named as E. Graph container was identical with Linkurious, so it meant that the Ogma was taking all the space available on the screen.

```
<meta charset="utf-8">
<script src="ogma.min.js"></script>
<style>
#graph-container { top: 0; bottom: 0; left: 0; right: 0; position: absolute; margin: 0; overflow: hidden; }
</style>
<div id="graph-container"></div>

<script>
'use strict';

function randomGraph(N, E) {
var g = {nodes:[], edges:[]};

for (var i = 0; i < N; i++) {
g.nodes.push({
id: 'n' + i,
text: 'Node ' + i,
x: Math.random() * 100,
y: Math.random() * 100
```

Figure 33. First impressions of demo-neighborhood.html code

Function randomGraph, shown in Figure 33, used familiar elements from the previous Linkurious tests. That existing code still did not work automatically with Ogma. This time reading the JSON into an array had to be done with the reformatted XHR code. Figure 34 shows that the synchronization setting for opening the JSON file had to be set to the value “false”. This change let the file load into an array before the browser drew the rest of the visualization.

```
<script>
'use strict';

var getJSON = function(url, successHandler, errorHandler) {
        var xhr = typeof XMLHttpRequest != 'undefined'
                ? new XMLHttpRequest()
                : new ActiveXObject('Microsoft.XMLHTTP');
        xhr.open('get', url, false);
        xhr.onreadystatechange = function() {
                var status;
                var data;

                if (xhr.readyState == 4) { //
                        status = xhr.status;
                        if (status == 200) {
                                data = JSON.parse(xhr.responseText);
                                successHandler && successHandler(data);
                        } else {
                                errorHandler && errorHandler(status);
                        }
                }
        };
        xhr.send();
};
```

Figure 34. Xhr request to get the response before the rest of the page

Loading the JSON file to an array before the rest of the site would have been possible with the Callback function as well. Callback function would have been a more sophisticated way to receive the data. Without Callback the page performs multiple functions simultaneously (Daggett, 2013). For our purpose the false setting was still sufficient.

Then it was time to launch the first Ogma test with the data. The result for 1000 nodes was 3.14 seconds which was surprisingly slower than with Linkurious. Immediate reload of the cached page came in 2.06 seconds which was considerably slower than the Linkurious. 1000 nodes on the screen did not look very user-friendly either. As shown on the left-hand side of Figure 35, the gray node ball and the street address texts were just too large with Ogma's default settings.

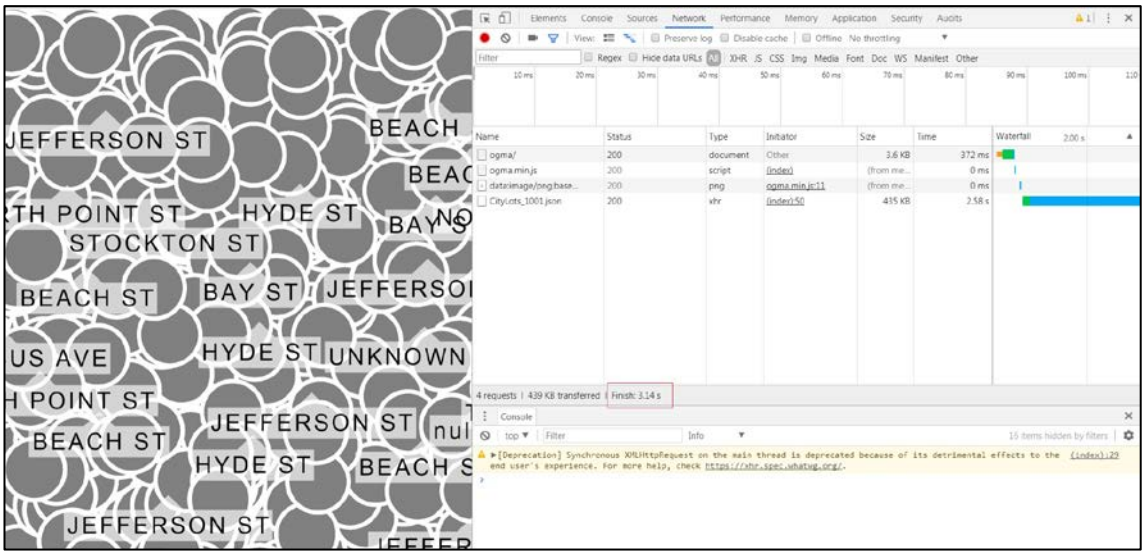


Figure 35. First test with the modified Ogma

The Ogma’s graphical settings had to be opened under investigation to make the user interface more effective. The Neighborhood highlight example did not directly show how the node size could be changed so the Ogman's web site was visited for help.

Ogma's home pages provided nicely structured examples and it was a response to the problems. The size of the nodes was defined within the drawing loop with the help of the “size” command. The engine accepted decimals, as long as the dot was used as a separator. Value 0.5 was tested to be the perfect size for further tests. The results can be seen in Figure 36.

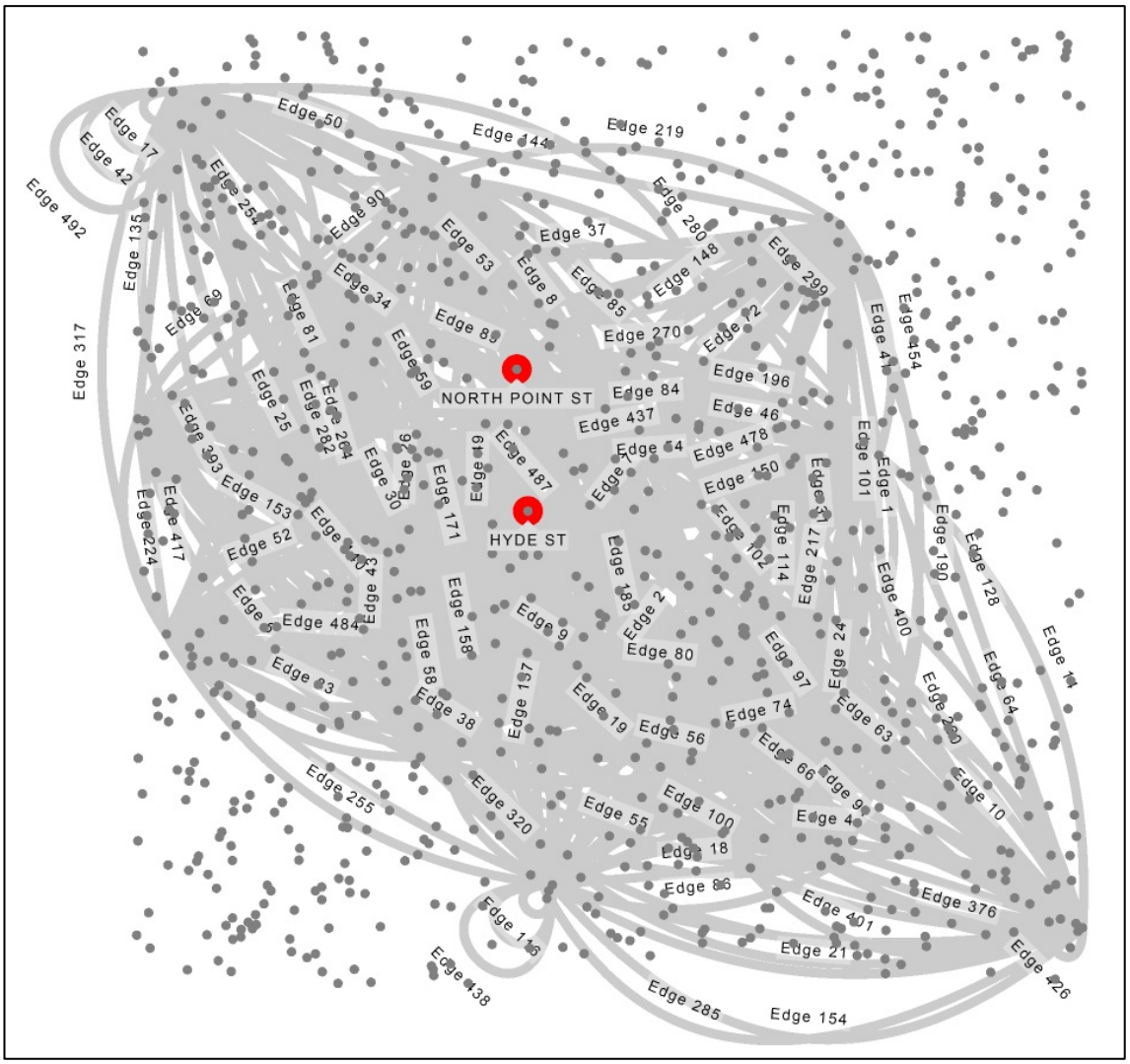


Figure 36. Ogma with 1000 city lots and smaller nodes

Figure 36 also shows the next problem. The nodes have been nicely reduced but the edges were too thick. They captured a huge part of the screen. Thus, the size command was again used and the edges were reduced to 0.1. The highlights of the nodes were decreased to 1 in the same round as seen in Figure 37. The nodes were highlighted in the example with the help of the mouseover effect.

```
function randomGraph(N, E) {
  var g = {nodes:[], edges:[]};

  for (var i = 0; i < N; i++) {
    g.nodes.push({
      id: 'n' + i,
      text: 'Node ' + i,
      x: Math.random() * 100,
      y: Math.random() * 100,
      size: 0.5
    });
  }

  for (var i = 0; i < E; i++) {
    g.edges.push({
      id: 'e' + i,
      text: 'Edge ' + i,
      source: 'n' + (Math.random() * N | 0),
      target: 'n' + (Math.random() * N | 0),
      size: 0.1
    });
  }

  return g;
}
```

Figure 37. Modifying the size of the nodes and edges

The test was started again with new settings, Ogma started to meet the expectations. The interface was beautiful, the navigation was smooth and the page was loaded in 788 milliseconds. Reducing the size of the nodes, edges, and highlight effects were in a crucial role in improving performance. Reloading the page generated 400 to 500 millisecond results with cache. The results are shown in Figure 38.

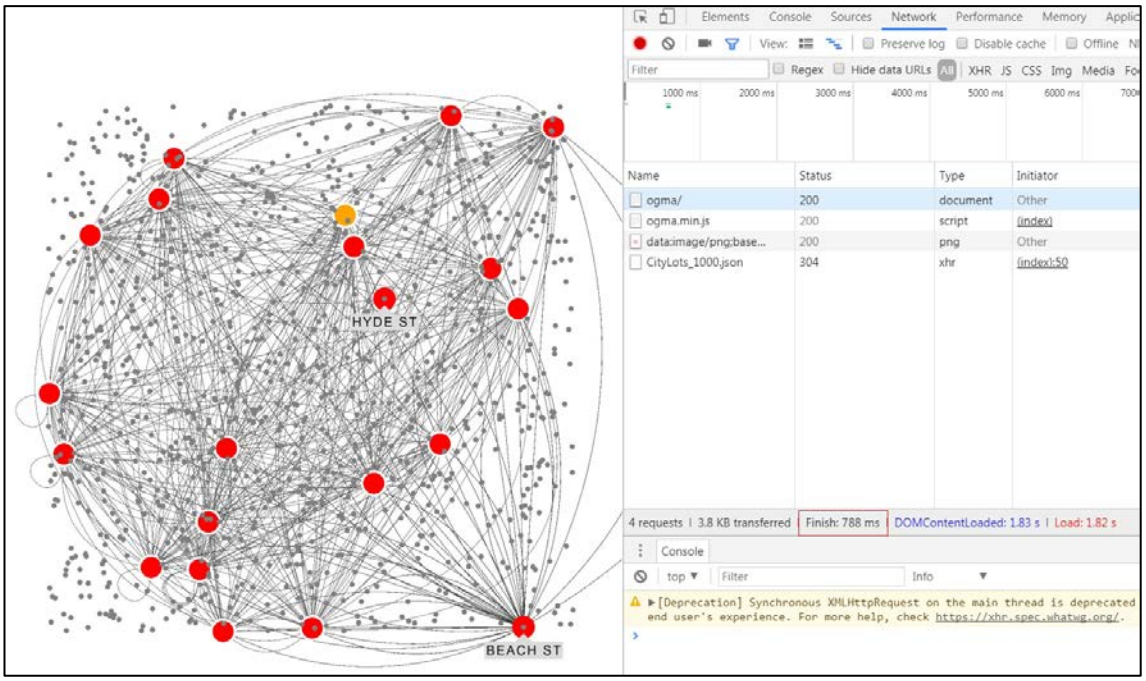


Figure 38. Ogma with the 1000 city lots and tweaked user interface

The material with the thousand rows gave some promising results with Ogma. It was time to move on with the tests. For comparison, the same data files were used with Ogma as with Linkurious. The results of the Ogma tests are shown in Table 2:

Table 2. Measurement results of Ogma. (* results from 50 000 and 100 000 rows are produced after the modifications)

The amount of City Lots	**Time (seconds)**	**User interface**
1 000	0,788	Working very smoothly
5 000	3,32	Working smoothly
10 000	4,31	Working well
25 000	8,62	Getting slowly
50 000	19,70	Working smoothly*
100 000	46,65	Working smoothly*

When the 50,000 rows of data was tested, JSON file was loaded quickly but the visualization just flashed on the screen and disappeared. The browser reported that it came across with a "CONTEXT_LOST_WEBGL" error. These following tips in Ogma's home

site were found during the investigation. These should increase the performance of Ogma (SAS, 2017):

1. A single character takes memory as much as one node. For example, the text "STREET" takes memory as much as six nodes.
2. With the large visualizations there is a setting called "render::webGLAntiAliasingsetting". Setting this from “automatic” to "none" makes the visualization a little bit rough, but performance should be better.
3. Reducing the size of the nodes and zooming out from the visualization will make the user interface easier to navigate.

The third tip was already implemented in the early stages of the Ogma tests, but the other two tips were helpful to in further investigations. The WebGL setting was founded inside the Ogma JavaScript library code. Settings "none" and "native" were tested, but neither of those changes did not have any effect. The WebGL error was still present.

The first tip on the list was tried next and the texts were completely removed from the nodes and the edges. WebGL setting was left to "none". This move helped the visualization remain on the screen and WebGL error did not exist anymore. The size of a single node (0,5) was still too large on the screen. The user interface looked nice but it was impossible to navigate smoothly because of the node size. The node size was reduced to 0,1 and the interface became clear again. The finished output is shown in Figure 39.

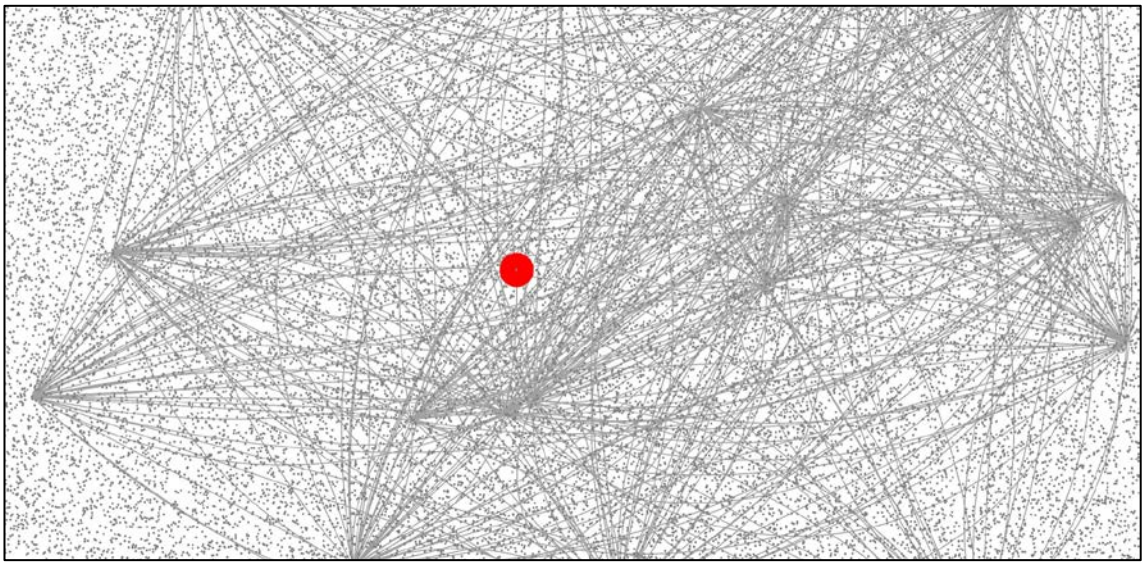


Figure 39. Ogma test with 50 000 rows, modified library and texts removed

Finally the data of 100 000 rows was placed into the test. The results were quite the same as with 50 000 rows. As indicated in Figure 40, data was loaded in less than one minute and the interface worked really smoothly.

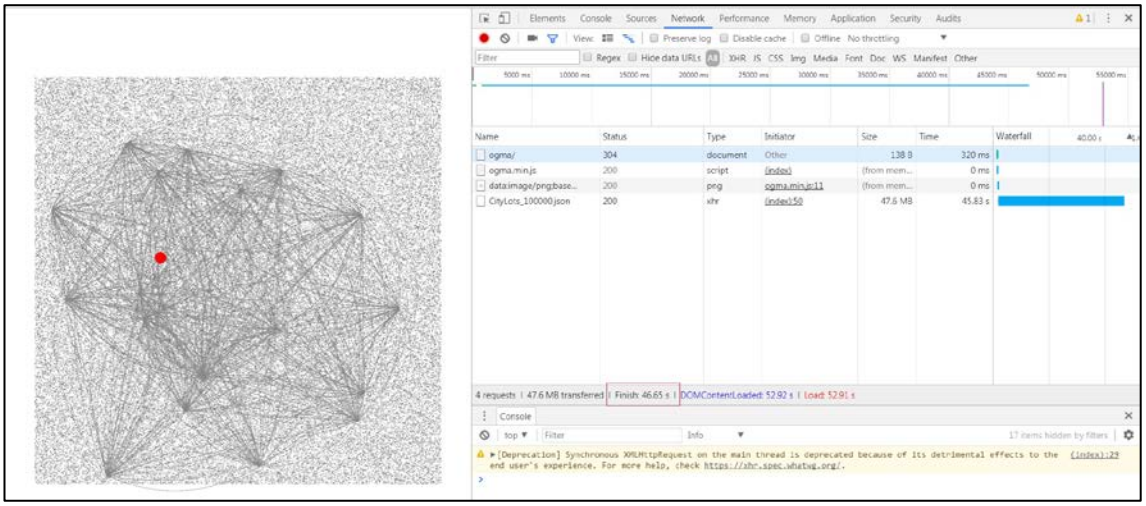


Figure 40. Ogma final test with 100 000 rows

Visualization was clear despite the lack of the texts in the nodes. At the end of the day, Ogma was able to bring 100,000 nodes to the screen as promised in the company's presentation material.

6 Discussion and Conclusions

In this final thesis, the modern JavaScript visualization libraries were investigated against today's performance challenges. From the hundreds of different libraries, the four best ones were selected and tested with massive data. In addition to the performance, the tests also paid attention to the interactivity and usability of the visualization. Before the tests, the history of interactive visualization libraries was also studied.

As a whole, the result of the study was both expected and positive. Things went all the way rather easily and the worst scenarios were avoided. The story of the thesis almost built itself because of the evolution of the Sigma libraries. The people behind the best data visualization engines were almost the same all the time. The story had a clear time frame. It is relieving to see that the developers have been active in a couple of year’s periods. New products have been steadily coming up every two or three years.

The development of the Interactive Data Visualization libraries seems to be in good hands. Many of the success stories are based on Open source code and Open data. This encourages communal development and demonstrates that it works for a common good. The worldwide use of Linkurious in the data leak cases shows the importance of the social relationships.

Ogma was published at the end of the year 2016 and it fulfilled the expectations that the present study was all about. There was a need for a gorgeous data visualization library for a massive piece of information. Such a library was found, as Ogma really had the ability to handle as much as 100,000 nodes. For the first time in the history there is an effective interactive data visualization library available.

6.1 Capabilities of Modern JavaScript Libraries

Exploring the JavaScript libraries revealed a lot of positive things. Especially Ogma brought some hope for the whole scene. It was a very nice package for both the developers and the users. Ogma's user friendliness can lower the threshold for developing new Linkurious innovations among non-programmers as well. Meanwhile D3.js still holds the top spot in the active community life but the situations can change quickly.

If the user happened to have his or her own JSON file, uploading the file directly to Ogma was possible. After that the file was browsable in Ogma's own ready-to-use platform. This interface was called Ogma Showcase and it was a fairly good extra service from them. The JSON file, however had to have a specific structure for the platform to accept it. Therefore the user needs at least average computing skills to convert the file to the required format (SAS, 2017).

All of the JavaScript libraries tested here were made quite easy as far as the learning curve is concerned. It seemed that the simplicity of deployment is a huge benefit in this scene as well. Flash and Java based interfaces showed that the willingness to install separate plugins into the browsers was very low. It was delightful to see in the tests that all the modern browsers worked with visualization libraries without any kind of tuning.

An eye-catching aspect was that the Linkurious had seamless co-operation with Neo4j database. Both of their philosophies relied heavily on the Open source world. Neo4j has not really any considerable competitor in sight at the moment. Graphene DB, a graph database hosting company, had an interview with Jean Villedieu at December 2015. Villedieu was the Chief Marketing Officer of Linkurious and he said the words which influenced also this Master's Thesis:

> The world is already structured as networks it can be social links, transactions, the way ideas spread. These are networks. It's a new way to present and think about information, which can empower you to make smarter decisions. I just saw a huge potential for this technology. (Villedieu, 2015).

As the thesis progressed, it became very clear that there are a lot of different online editors in the world of data visualization. For example, there were online editors for SVG, Canvas and Neo4j. Most of them were doing serious business with the products. Generally they worked so that the user was given the opportunity to make a simple image or visualization for free. Behind the paywall were then such materials as ready-made templates. In any case, data visualization has more and more relocated from the desktop to the Internet and cloud services.

6.2 Development Needs of Interactive Data Visualization

For the future, there are many things to improve. The weakest link of the visualization engines is the so-called lack of the visualization continuity. When a user browses and analyzes the material, it would be very important that clicking the node would take the

user forward in the visualization. This important feature is missing on most of the engines. Even the developers of Linkurious have not implemented it fully.

Another major disappointment in visualization engines was the lack of user-friendliness with the JSON data handling. People's own JSON file structures can be completely different from what visualization engines will take by default. The user then has normally two options:

- Modify the structure of his own JSON file
- Modify the JSON parsing function inside the loop

Both options needs a little bit a work. A small editor or wizard in the visualization engines would be a better option. It could ask the user what fields in the JSON file the user would like to see in the visualization.

Most of the biggest interactive data visualization libraries look like they have been designed by the same artist. In the world of infographic, there have been many great static images, but it seems that the artistic talents has not yet found the world of data visualization. If someone made a modern infographic style data visualization user interface, it could have a good future ahead.

The size of the screens is one problem in the world of data visualization. Computers, laptops and tablets are basically not designed for a full-scale analysis. Greater data volumes would be easier to handle on larger screens. The biggest touch screens have not yet been taken into use with their full capacity.

Chapter 5 introduced the final performance tests with Linkurious and Ogma libraries. One of the lessons learned was the fact that the browsers did not return their zoom settings to default. So if the zoom effect was maximized, the navigation worked with a slower pace. If the user has a mouse and roll function between the buttons, then this is not a big problem. Also with the tablets this problem can be easily avoided.

One of the tests with Ogma library was paused for a moment because there was no chance to fix the visualization from the JavaScript code inside the HTML pages. Instead, the changes had to be made inside the core library. In general, actions of this kind are

not very encouraging to non-programmers. Even the talented developers do not want to touch the manufacturer’s ready-made libraries.

6.3 Validity

The results of the tests were in line with the promises of library manufacturers. They were also in line with the discussions on the Internet bulletin boards. For D3.js, 10 000 nodes started to be too much and Linkurious slowed down considerably with the same material. Ogma survived with 100 000 nodes although not with the full details.

If using a local web server, the results would have been extremely good. But in the tests it was wanted to see the visualizations in the terms of everyday life. That is why an external web server was used. The tests can be replayed as long as the Internet Service Provider gives the connection of 100/10 Mbps. If the data visualization engines have to be presented to the general public at a trade show, a local web server is a good choice.

6.4 Future

It is exciting to see in the future what kind of competition there will be between the data visualization engines. Linkurious's products clearly dominate the field now. Their tempo is just increasing with the Ogma. On the other hand, the thesis also revealed that Ogma's roots are still deep in the Sigma.js library. The improvement of performance has therefore become from logical changes, not through new technical innovations.

The history of web data visualization techniques has become from Java to Flash and from Flash to JavaScript. Is there a new technology coming around the corner or does the JavaScript libraries still keep their engines running? Time will show what the strength of the communities among the JavaScript libraries is. The D3.js is still very popular so they might also introduce the next big thing in the scene.

In the future, data visualization might be functioning more and more in real-time. We have not yet seen revolutionary user interfaces in this area, but it is only a matter of time when they arrive. There is a need for at least real-time monitoring, tracing and measuring applications (Crooks, 2011).

References

Bastian, M., 2016. *A close look at the Gephi user community.* [Online]
Available at: https://gephi.wordpress.com/2016/02/06/a-close-look-at-the-gephi-user-community/
[Accessed 29 July 2017].

Bostock, M., 2017. *Data-Driven Documents.* [Online]
Available at: https://d3js.org/
[Accessed 25 July 2017].

Bruggen, R. V., 2014. *Learning Neo4j.* 1st ed. s.l.:Packt Publishing.

Campesato, O., 2003. *Fundamentals of SVG Programming.* 1st ed. s.l.:Charles River Media.

Carey, S., 2017. *NASA turns to graph database to help researchers and prove its worth to the taxpayer.* [Online]
Available at: http://www.computerworlduk.com/data/nasa-turns-graph-database-prove-its-research-is-benefitting-taxpayer-3653015/
[Accessed 26 July 2017].

Chun-houh Chen, W. H. A. U., 2008. Visualization for Genetic Network Reconstruction. In: C. G. Grace S. Shieh, ed. *Handbook of Data Visualization.* s.l.:Springer, p. 871.

City, S. F., 2017. *SF Open Data.* [Online]
Available at: https://data.sfgov.org/Geographic-Locations-and-Boundaries/City-Lots/45et-ht7c
[Accessed 18 July 2017].

Crooks, R., 2011. *3 Trends That Will Define The Future Of Infographics.* [Online]
Available at: https://www.fastcodesign.com/1665029/3-trends-that-will-define-the-future-of-infographics
[Accessed 28 July 2017].

D3.js, 2017. *d3 Gallery.* [Online]
Available at: https://github.com/d3/d3/wiki/gallery
[Accessed 29 July 2017].

Daggett, M. E., 2013. *Expert JavaScript.* 1st ed. s.l.:Apress.

Dormehl, L., 2014. The Five Best Libraries For Building Data Visualizations. *Fast Company*, 28 April.

Francisco, C. a. C. o. S., 2017. *DataSF.* [Online]
Available at: https://data.sfgov.org/
[Accessed 16 July 2017].

Friendly, M., 2008. *Handbook of Data Visualization.* 1st ed. s.l.:Springer.

Gartner, 2016. *Gartner Survey Reveals Investment in Big Data Is Up but Fewer Organizations Plan to Invest.* [Online]

Available at: http://www.gartner.com/newsroom/id/3466117
[Accessed 27 July 2017].

Glenn J. Myatt, W. P. J., 2011. *Making Sense of Data III: A Practical Guide to Designing Interactive Data Visualizations.* 1st ed. s.l.:Wiley.

Hawkes, R., 2011. *Foundation HTML5 Canvas.* 1st ed. s.l.:Apress.

Heer, J., 2008. *Flare Sample Applications.* [Online]
Available at: http://flare.prefuse.org/apps/index
[Accessed 24 July 2017].

Heymann, S., 2016. *Leaked documents.* [Online]
Available at: https://linkurio.us/blog/panama-papers-how-linkurious-enables-icij-to-investigate-the-massive-mossack-fonseca-leaks/
[Accessed 23 07 2017].

Heymann, S., 2017. *Linkurious Blog.* [Online]
Available at: https://linkurio.us/blog/ogma-js-library-large-scale-graph-visualization/
[Accessed 23 July 2017].

Hohenadel, K., 2014. *How Doodles and Sketches Become Gorgeous Infographics.* [Online]
Available at:
http://www.slate.com/blogs/the_eye/2014/09/26/infographic_designers_sketchbooks_by_steven_heller_and_rick_landers_reveals.html
[Accessed 28 July 2017].

Home, L., 2017. *Linkurious Home Site.* [Online]
Available at: https://linkurio.us/
[Accessed 23 July 2017].

Jacomy, A., 2012. *Sigma.js Home.* [Online]
Available at: http://sigmajs.org/
[Accessed 26 July 2017].

Koppal, T., 2017. *Trends in Data Visualization.* [Online]
Available at: http://www.labmanager.com/ask-the-expert/2017/05/trends-in-data-visualization#.WXsTDISLSpo
[Accessed 28 July 2017].

Krum, R., 2013. In: *Cool Infographics : Effective Communication with Data Visualization and Design.* s.l.:Wiley, pp. 11-12.

Labs, N., 2012. *Sigma.Js – The Amazing New Online Graph Visualization Tool.* [Online]
Available at: http://noduslabs.com/radar/sigma-js-online-graph-visualization-tool/
[Accessed 26 July 2017].

Lab, U. B. V., 2008. *Flare Home Site.* [Online]
Available at: http://flare.prefuse.org/
[Accessed 24 July 2017].

Lankow, J., Crooks, R. & Ritchie, J., 2012. In: *Infographics: The Power of Visual Storytelling.* s.l.:John Wiley & Sons 2012, p. 14.

Linkurio.us, 2017. *Linkurious Enterprise Product Page.* [Online]
Available at: https://linkurio.us/product/
[Accessed 23 July 2017].

Linkurious.js, 2017. *Linkurious.js Home.* [Online]
Available at: https://github.com/Linkurious/linkurious.js/
[Accessed 23 July 2017].

Marques, J. P., 2014. *Using Sigma for Drawing Graphs.* [Online]
Available at: https://www.infoq.com/news/2014/03/sigma-javascript-graph-drawing
[Accessed 26 July 2017].

Marr, B., 2015. *Big Data.* 1st ed. s.l.:Wiley.

Michael Bostock, J. H., 2009. *Protovis: A Graphical Toolkit for Visualization,* s.l.: IEEE Trans. Visualization & Comp. Graphics.

MongoDB, 2017. *Big Data Explained.* [Online]
Available at: https://www.mongodb.com/big-data-explained
[Accessed 29 July 2017].

Murray, S., 2013. *Interactive Data Visualization for the Web: An Introduction to Designing with D3.* 1st ed. s.l.:O'Reilly Media.

Neo4j, I., 2017. *Using Neo4j in Open Source Software.* [Online]
Available at: https://neo4j.com/open-source/
[Accessed 28 July 2017].

Nylen, J., 2016. *D3 Process Map.* [Online]
Available at: https://github.com/nylen/d3-process-map
[Accessed 25 July 2017].

Panama Papers: How the French start-up Linkurious helped to make it all happen. 2016. [Film] Directed by J. Barrere & W. Hilderbrandt. France: France 24.

Perna, M. A., 2016. *Canvas vs. SVG: Choosing the Right Tool for the Job.* [Online]
Available at: https://www.sitepoint.com/canvas-vs-svg-choosing-the-right-tool-for-the-job/
[Accessed 28 July 2017].

Portal, E. D., 2015. *Creating Value through Open Data,* s.l.: European Commission.

Roy A. Adkins, L. A., 2002. In: *The Little Book of Egyptian Hieroglyphs.* s.l.:Hodder Mobius, pp. 13-14.

SAS, L., 2017. *Linkurious Examples.* [Online]
Available at: https://rawgit.com/Linkurious/linkurious.js/develop/examples/index.html
[Accessed 29 July 2017].

SAS, L., 2017. *Neighborhood Highlight.* [Online]
Available at: https://doc.linkurio.us/ogma/latest/examples/demo-neighborhood.html
[Accessed 23 July 2017].

SAS, L., 2017. *Ogma API render settings.* [Online]
Available at: https://doc.linkurio.us/ogma/1.5.3/api.html#render-settings
[Accessed 23 July 2017].

SAS, L., 2017. *Ogma Showcase.* [Online]
Available at: http://linkurious.github.io/ogma-showcase/
[Accessed 23 July 2017].

Shane, S., 2016. Panama Papers May Inspire More Big Leaks, if Not Reform. *New York Times: World,* Volume I, p. 1.

Skau, D., 2013. *Visually Blog.* [Online]
Available at: https://visual.ly/blog/why-d3-js-is-so-great-for-data-visualization/
[Accessed 25 July 2017].

Smiciklas, M., 2012. *The Power of Infographics.* 1st ed. s.l.:Que.

Smith, S., 2015. *The Disadvantages of Scalable Vector Graphics.* [Online]
Available at: https://www.techwalla.com/articles/the-disadvantages-of-scalable-vector-graphics
[Accessed 25 July 2017].

Sourceforge.net, 2006. *Prefuse Example Application.* [Online]
Available at: http://prefuse.org/doc/manual/introduction/example/
[Accessed 24 July 2017].

St-Maurice, I., 2015. *Three BIG reasons companies are failing with Big Data Analytics.* [Online]
Available at: https://www.linkedin.com/pulse/three-big-reasons-companies-failing-data-analytics-ian-st-maurice
[Accessed 27 July 2017].

Thomas, S. A., 2015. *Data Visualization with JavaScript.* 1st ed. s.l.:No Starch Press.

Wijaya, H., 2016. *25 Best Javascript Chart & Graph Libraries & Tools.* [Online]
Available at: http://bashooka.com/coding/25-best-javascript-chart-graph-libraries-tools/
[Accessed 31 August 2017].

Villedieu, J., 2015. *Graphene DB.* [Online]
Available at: https://www.graphenedb.com/blog/2015/12/28/interview-with-jean-villedieu-of-linkurious/
[Accessed 23 July 2017].

Villedieu, J., 2015. *How NASA Experiments with Knowledge Discovery.* [Online]
Available at: https://linkurio.us/blog/how-nasa-experiments-with-knowledge-discovery/
[Accessed 26 July 2017].

Yuk, M., 2014. *Data Visualization For Dummies.* 1st ed. s.l.:For Dummies.