

Similarity Map for RAW Graphs

Manuel Schweiger, Reinhold Seiss, Maximilian Tauß, Stefan Tscheppe

Information Visualisation course at
Institute of Interactive Systems and Data Science (ISDS),
Graz University of Technology
A-8010 Graz, Austria

2 July 2018

Abstract

Information Visualisation is a growing field and gains popularity as data aggregation becomes easier and big data more accessible. Many people working in different areas rely on Information Visualisation tools and editors in order to present their findings and support their argumentations. One of such tools is RAW Graphs. It provides a manifold of different visualisation techniques.

We will show what a Similarity Map is, which algorithms are needed for such a visualisation technique and how to extend RAW Graphs to also feature this technique.

© Copyright 2018 by the author(s), except as otherwise noted.

This work is placed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence.

Contents

Contents	i
List of Figures	iii
List of Listings	v
1 Introduction	1
2 RAW Graphs	3
2.1 About	3
2.2 Workflow	5
2.3 Extendability	8
2.4 Evaluation of RAW Graphs	8
3 Similarity Map	9
3.1 About	9
3.2 Technical Details	10
4 Algorithm	11
4.1 Force-Directed Placement	11
4.2 t-Distributed Stochastic Neighbor Embedding	12
5 Extending RAW Graphs	13
5.1 Dependencies	13
5.2 Setup	13
5.3 Project Structure	13
5.3.1 The <code>index.html</code> File	14
5.3.2 The <code>js/controller.js</code> File	14
5.3.3 The <code>data</code> Folder	14
5.3.4 The <code>charts</code> Folder	14
5.3.5 The <code>imgs</code> Folder	14
5.4 Adding a New Chart	14
5.5 Drawbacks of RAW Graphs	16
5.5.1 Dynamic optional Input Fields	16
5.5.2 The <code>draw()</code> Function	17
5.5.3 SVG Export	17
6 Concluding Remarks	19
Bibliography	21

List of Figures

2.1	RAW Graphs: Built-In Chart Types	4
2.2	RAW Graphs: Uploading Data Set	5
2.3	RAW Graphs: Choosing A Chart	6
2.4	RAW Graphs: Mapping data to specific axes	6
2.5	RAW Graphs: Customizing The Chart	7
2.6	RAW Graphs: The Finished Barchart	7
2.7	RAW Graphs: Downloading The Barchart	8
3.1	Similarity Map: Purchasing Power Different Cities	9

List of Listings

4.1	Algorithms: Cosine Similarity Implementation	11
4.2	Algorithms: Simple Force Directed Placement Implementation	11
5.1	RAW Graphs Framework: Adding A New Chart	14
5.2	RAW Graphs Framework: Adding Code	14
5.3	RAW Graphs Framework: Defining Model	15
5.4	RAW Graphs Framework: Mapping Data	15
5.5	RAW Graphs Framework: Chart Description	15
5.6	RAW Graphs Framework: Defining Parameters	15
5.7	RAW Graphs Framework: The Draw Function	16

Chapter 1

Introduction

Information Visualisation editors are used to produce charts representing a specific set of data. Typically the more complex the data is, the more sophisticated the editor needs to be in order to output a meaningful representation of the given data. RAW Graphs is a simple and easy to use visualisation editor, only limited by the built-in chart types it offers.

In this project we show how to further extend RAW Graphs by a Similarity Map. We discuss the implemented algorithms needed to implement a Similarity Map and how to integrate that into the existing framework of RAW Graphs.

Chapter 2

RAW Graphs

First we take a look at RAW Graphs [Mauri et al. [2017]]. As similar editors it is very easy to use, includes a self-explanatory user interface and offers a predefined range of chart types. Overall it is very beginner-friendly and one can produce charts very quickly.

Due to being streamlined and very polished from beginning to end, RAW Graphs cuts down on customisability. For example axis labels or other label like color descriptions can not be added in the editor itself and have to be added by the user in a post-processing step if needed.

2.1 About

The editor, more specifically a hosted web application of the editor, can be found at the website [*RAW Graphs Website* [2018]]. RAW Graphs is an open source data visualisation framework. Its goal is to make the visualisation of complex data easy and accessible to everyone [*RAW Graphs Website* [2018],About]. Although it is self-hosted, RAW specifically states that any uploaded data stays on your machine and is not uploaded to any of their servers.

The framework is based on the popular JavaScript data visualisation library D3 [*RAW Graphs Github* [2018]]. Initially it was developed by the DensityDesign Research Lab and Calibro and after a couple of years ContactLab got involved as well.

Currently it comes with visualisation techniques like contour plots, convex hull, hexagonal binning, scatter plot, voronoi tessellation, beeswarm plot, sunburst, treemap, parallel coordinates, see Figure 2.1, and many more – 21 so far.

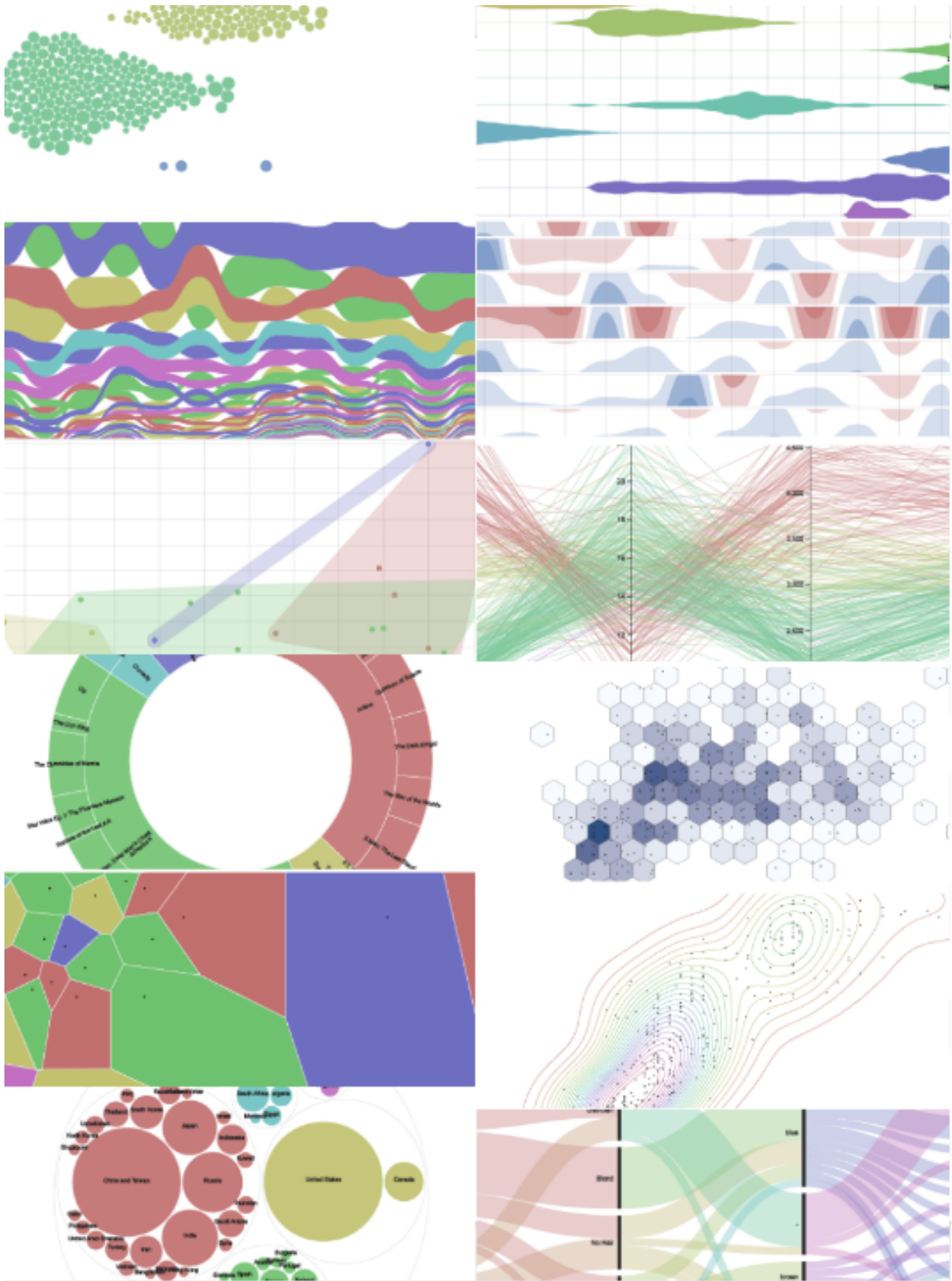


Figure 2.1: This shows some of the visualisation techniques featured in RAW Graphs.

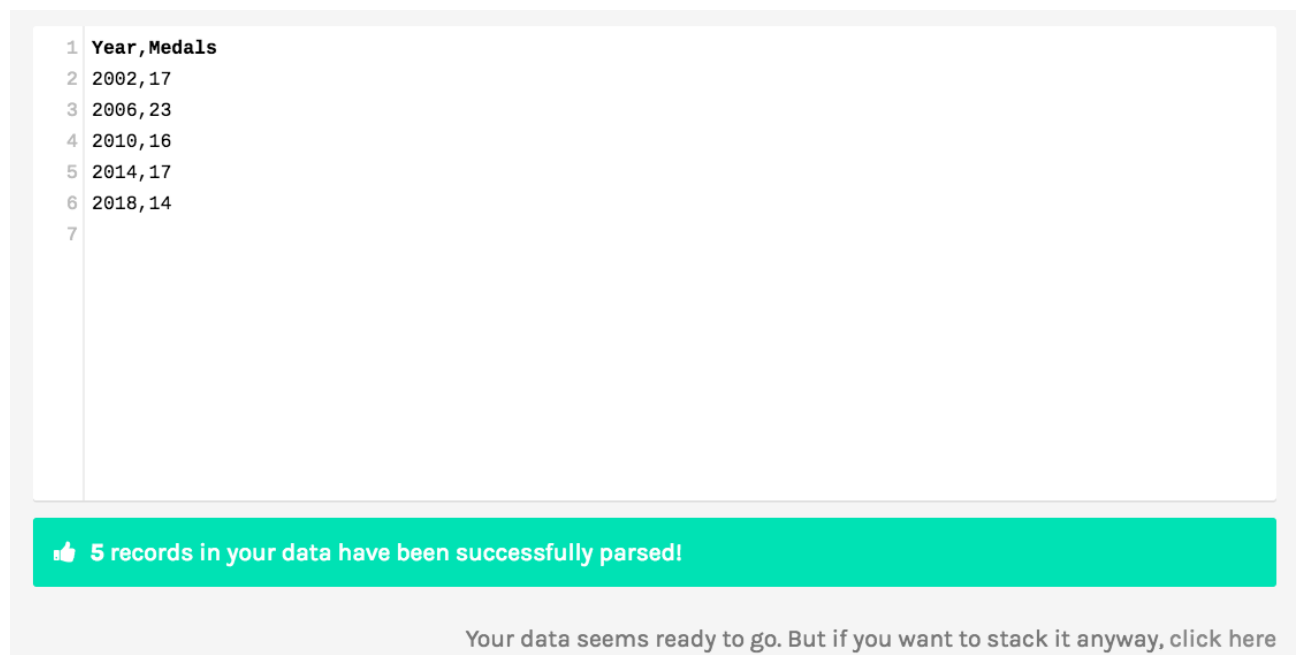
More information about RAW Graphs can be found on the webpage of its inventors.

2.2 Workflow

We will illustrate the workflow of RAW Graphs on a practical example. Usually RAW Graphs is used as a link in-between any spreadsheet software like Microsoft Excel and any vector graphics editor like Adobe Illustrator. This will become more clear as we will explain how to create a simple barchart, step by step.

The dataset we are going to use for the barchart example is about the medals Austria won in the Winter Olympics of the last 5 years. In RAW Graphs the creation of a chart entails the following steps:

1. Upload the dataset and make adjustments if needed (see Figure 2.2).
2. Choose one of the predefined chart types (see Figure 2.3). Note that there always is a description of the highlighted chart type.
3. Link and map your given data to the specific axes available (see Figure 2.4). Depending on the chosen chart type this view might be different.
4. Customize the chosen chart (see Figure 2.5).
5. Double check the created chart (see Figure 2.6) and make further adjustments if necessary.
6. Download your chart (see Figure 2.7) by using one of the available options. Note that you also can directly copy the embed SVG code.



The screenshot displays a data table with the following content:

1	Year, Medals
2	2002, 17
3	2006, 23
4	2010, 16
5	2014, 17
6	2018, 14
7	

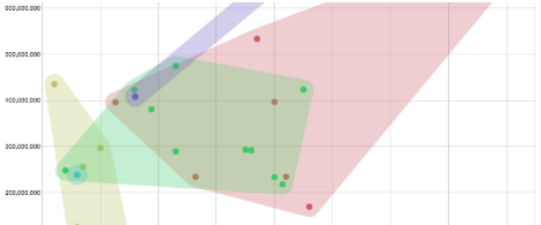
Below the table, a green notification bar states: "👍 5 records in your data have been successfully parsed!". At the bottom, a link is provided: "Your data seems ready to go. But if you want to stack it anyway, click here".

Figure 2.2: After we have uploaded our example data set.

Choose a Chart


Convex Hull

Dispersion




In mathematics, the convex hull is the smallest convex shape containing a set of points. Applied to a scatterplot, it is useful to identify points belonging to the same category.

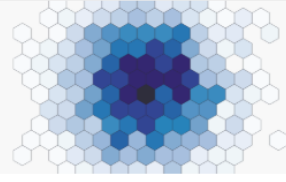
Based on <http://blocks.org/mbostock/4341699>



Convex Hull
Dispersion



Delaunay Triangulation
Dispersion



Hexagonal Binning
Dispersion

Figure 2.3: Choosing A Chart

Map your Dimensions

Year number →

Medals number →

X Axis

Drag numbers, strings here

Year number ×

Height

Drag numbers here

Medals number ×

Figure 2.4: Mapping data to specific axes.

Customize your Visualization

Width
800

Height
600

Vertical Padding
0

Horizontal Padding
0,1

Use Same Scale

Color Scale
Ordinal (categories) ▾

Search...
null

#bf6969

Figure 2.5: Customizing The Chart

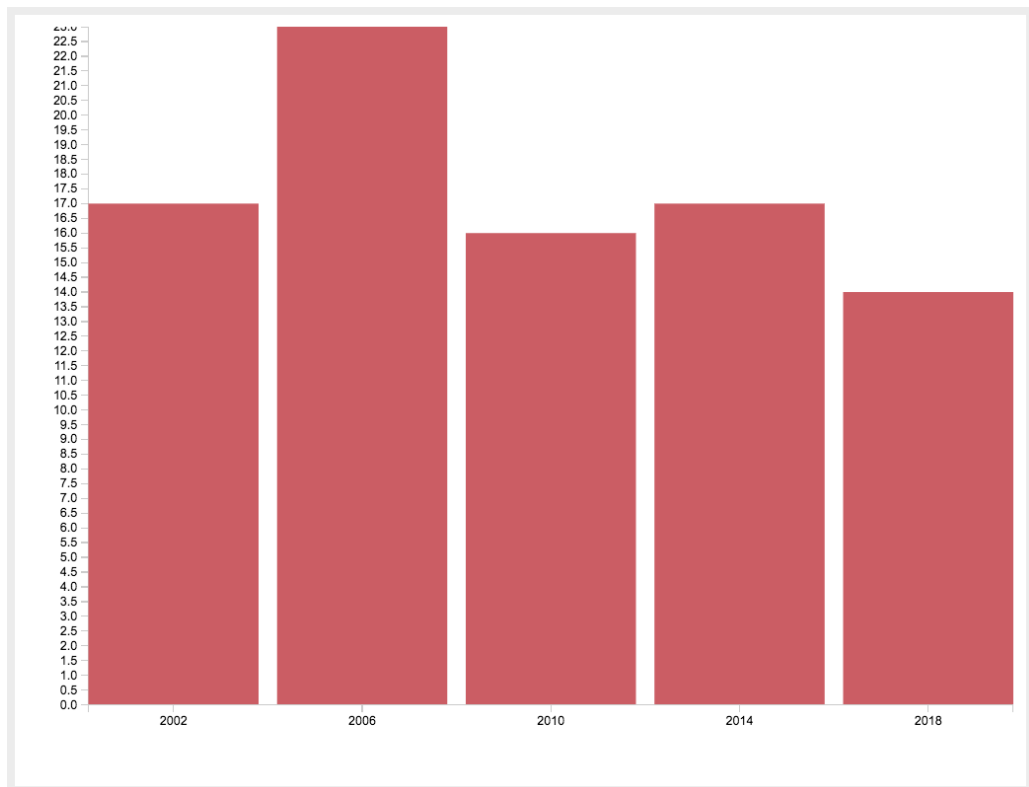


Figure 2.6: The Finished Barchart

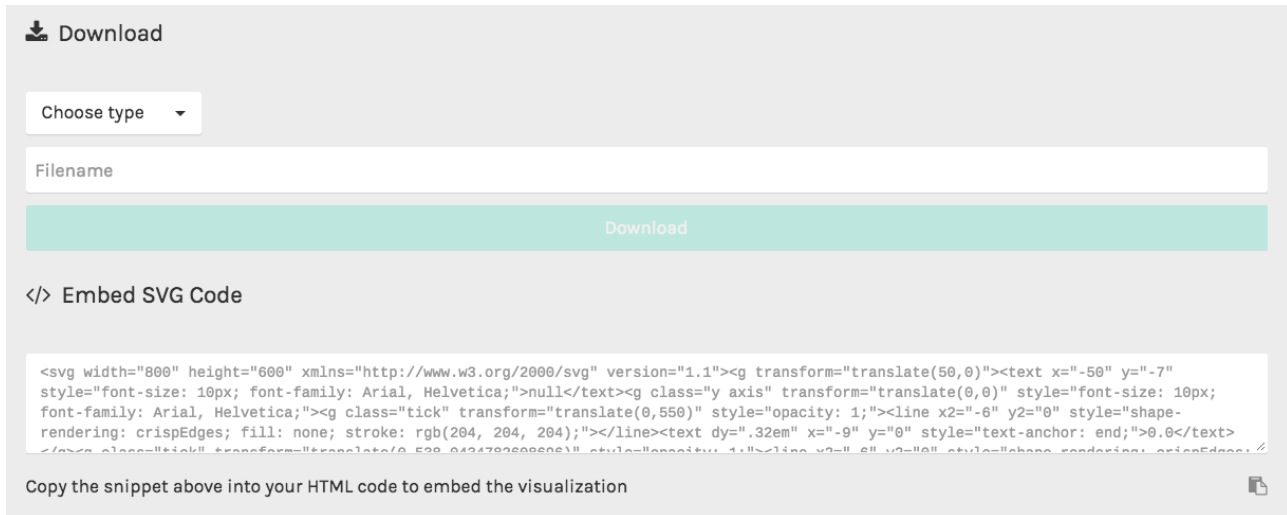


Figure 2.7: Downloading The Barchart

2.3 Extensibility

Note that if you have the required skills you can also fork the entire project on Github [[RAW Graphs Github \[2018\]](#)] and self host RAW Graphs or create custom chart types this way.

2.4 Evaluation of RAW Graphs

Here you can see our evaluation of RAW Graphs. It is a finished product and you can clearly see that the services it offers are highly polished and streamlined. Its user base is well defined and it cemented its existence into the heart of the data visualisation community, which wants to do just that – visualise data. Following is our evaluation of RAW Graphs:

1. *Platform*: hosted web application (uploaded data stays local)
2. *Pricing and license*: free under the Apache License 2.0
3. *Customisability*: low
4. *Range of chart types*: low, but extendable if you fork the Github project
5. *Usability*: very good
6. *Export formats*: SVG / PNG / JSON

Chapter 3

Similarity Map

In this chapter we will talk about what a Similarity Map is and how one can be created. There are different approaches and algorithms, each producing unique but similar results.

3.1 About

Similarity Maps are a visualisation technique used to project multi dimensional data onto a two dimensional plane. It shows the similarity between different feature points by the closeness or distance of the plotted points on the chart. See Figure 3.1 for an example.

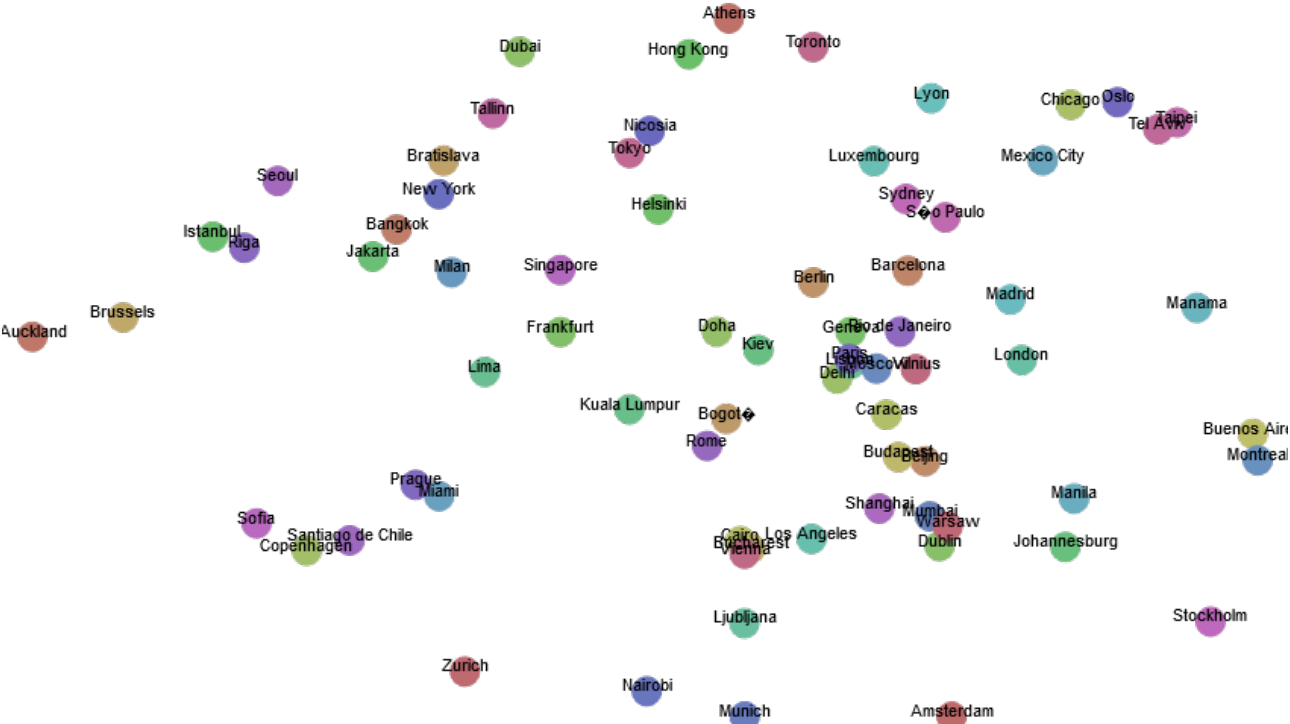


Figure 3.1: This Similarity Map shows the purchasing power per household on average of different cities.

3.2 Technical Details

Different algorithms used to create Similarity Maps are Force-Directed Placement [Eades [1984]], Principal Component Analysis [Abdi and Williams [2010]], Multi-Dimensional Scaling [Borg and Groenen [2005]], t-distributed stochastic neighbor embedding [van der Maaten and Hinton [2008]] and many more.

In our project we focused on Force-Directed Placement and t-distributed stochastic neighbor embedding (t-SNE). We implemented the first ourselves and used an existing library provided by the inventors of the algorithm for the second.

Note that some of the algorithms use random initializations and therefore are non-deterministic. This holds for Force-Directed Placement as well as t-SNE. With random initialization one will always get different results on the same dataset.

Chapter 4

Algorithm

In this chapter we will try to give a short overview of the two different algorithms that we used to calculate our similarity map. Especially we will take a look at the different pros and cons every approach offers.

4.1 Force-Directed Placement

The gist of the force directed placement is quite easy. By defining a set of forces, e.g. attraction between similar objects, over all the different points, a simplified physics system is created. After initializing with arbitrary positions this simulation is played out and therefore the positions are calculated. This process can be summarized in 3 essential steps:

1. Feature Extraction: As the name suggest the features of the given data points are extracted and saved to an appropriate matrix.
2. Similarity Calculation: Here the similarity or dissimilarity between the individual data points is calculated by an appropriate metric. In our case we decided to use the Cosine Similarity.

```
1 function similarity(vectorA, vectorB) {
2   var dotProduct = 0.0;
3   var normA = 0.0;
4   var normB = 0.0;
5   for (var i = 0; i < vectorA.length; i++) {
6     dotProduct += vectorA[i] * vectorB[i];
7     normA += Math.pow(vectorA[i], 2);
8     normB += Math.pow(vectorB[i], 2);
9   }
10  return dotProduct / (Math.sqrt(normA) * Math.sqrt(normB));
11 }
```

3. Force Directed Placement: This is the most important step. Here the forces are initialized according to the the values of the similarity matrix and then it will be iterated either until it is stable or the number of maximum iterations is reached. In our case we decided to use a very simple force model to avoid too complicated computations.

```
1 function calculateForce(i, data, similarityMatrix) {
```

```

2 |   var force = {x: 0, y: 0};
3 |   for (var j = 0; j < data.length; j++) {
4 |     if (i != j) {
5 |       var direction = {
6 |         x: data[i].x - data[j].x,
7 |         y: data[i].y - data[j].y
8 |       };
9 |       var normalizedDirection = normalize(direction);
10 |
11 |       force.x += normalizedDirection.x * similarityMatrix[i][j];
12 |       force.y += normalizedDirection.y * similarityMatrix[i][j];
13 |     }
14 |   }
15 |   return force;
16 | }

```

Problems, Pros and Cons. One of the biggest advantages of this approach was that it was fairly easy to implement and it can also be done with very little amount of code. The physic based approach also makes it quite intuitive to understand and to predict. Another benefit is that because of the iterative aspect this could be used for interactive visualization and the user can get a understanding on how the graph evolves. Unfortunately because of the static nature of RAW and our decision to use randomized starting points for the individual points we couldn't play into this strength of the method. This could have been somewhat mitigated by not initializing the starting points randomly but by calculating them deterministically by e.g. using PCA.

4.2 t-Distributed Stochastic Neighbor Embedding

This is a more sophisticated approach that is based on machine learning. It is also a quite new technique and has first been proposed in 2008 by Laurens van der Maaten and Geoffrey Hinton [van der Maaten and Hinton [2008]]. It is non linear and based on dimensionality reduction. It is compromised by two main stages:

1. First a probability distribution over the high dimensionality data is constructed in such a way that similar data points are very likely to be picked and dissimilar points are very unlikely.
2. A similar probability is then constructed over the points projected to the lower dimensional space. Then the Kullback-Leibler divergence between these two distributions is minimized.

Problems, Pros and Cons. Since this approach is by far more complicated then the Force-Directed Placement, it was out of the scope of this project for us to implement our own implementation. This may have saved us a lot of time, but it also reduces the amount of insight we could gather about this visualization technique. One of the biggest insights that we could gather was that the perplexity parameter very much defines the outcome of the algorithm, and that clustering is not reliable in t-Distributed Stochastic Neighbor Embedding as it does not preserve distances.

Chapter 5

Extending RAW Graphs

5.1 Dependencies

RAW Graphs is licensed under the Apache License 2.0 which means that everyone is allowed to extend the application as he likes. Additionally, RAW Graphs feature their own GitHub page including the source code, tutorials and a wiki. Therefore, it is rather easy to add additional visualisation techniques such as a Similarity Map. The following dependencies are given:

- Bower (1.8.4)
- D3.js (3.5.17)
- Angular (1.6.2)
- Bootstrap (3.3.6)
- Python (2.7.4) to host a local server

5.2 Setup

In order to extend RAW Graphs, one has to clone their repository from GitHub. This can be easily done by `git clone https://github.com/densitydesign/raw.git`. After the download process one can setup the project by simply executing `bower install` inside the project folder. RAW Graphs is now ready to run and can be hosted on a local server with the command `python -m http.server 4000` Using a simple browser like Firefox or Chrome, RAW Graphs can be used locally under `http://localhost:4000/`.

5.3 Project Structure

RAW Graphs comes with a very easy and intuitive project structure. To extend RAW Graphs with a new chart type the following files and folders are important:

- `index.html`
- `js/controller.js`
- `data`
- `charts`

- `imgs`

5.3.1 The `index.html` File

The file `index.html` describes the start of the RAW Graphs application. Script files which are needed are included and initialized as well as the chart types.

5.3.2 The `js/controller.js` File

This script file does a lot of background processing which should not be of interest if you simply want to add a new chart type. But in order to add additional sample data to RAW Graphs, which we did to simplify the workflow, `scope.samples` has to be extended inside this file.

5.3.3 The data Folder

The “data” folder includes all the sample data sets of RAW. To fasten up the debugging process we included our own sample data set which works smoothly with our similarity map. This step is of course optional since data sets can be easily uploaded if needed, it saves some time though to permanently include it. The format of the data set should be *CSV*.

5.3.4 The charts Folder

After adding the reference of a new chart type in `index.html`, RAW Graphs looks for this particular file inside the `charts` folder.

5.3.5 The `imgs` Folder

Since all chart types feature a thumbnail in the chart overview of RAW Graphs, an image is needed. All images are in *PNG* format and stored inside this folder.

5.4 Adding a New Chart

This is a simple tutorial which can also be found on <https://github.com/densitydesign/raw/wiki/Adding-New-Charts> and shows how to easily extend RAW Graphs with a simple scatter plot. Start by creating a new *JS* file inside the `charts` folder. Let’s call it `sample.js`. To tell RAW Graphs to include this chart type we add the following line to `index.html`:

```
1 <script src="charts/ sample.js"></script>
```

Now we can implement the chart by starting with:

```
1 (function() {  
2 // your code here...  
3 })();
```

This is a self-executing function which is recommended by RAW Graphs. Now we can define our model.

```

1 var model = raw.model();
2 var x = model.dimension()
3   .title("X Axis")
4   .types(Number);
5 var y = model.dimension()
6   .title("Y Axis")
7   .types(Number);

```

The method `Model.dimension()` tells RAW Graphs which properties we want to expose to the user. Since we implement a simple scatter plot, we want the user to map his data onto the X- and Y- axis. We can restrict the data type to numbers for each axis by adding the type `Number`.

Now we need to map the data which the user wants for his scatter plot axis to objects which we can use later to construct our scatter plot.

```

1 model.map(function(data) {
2   return data.map(function(d) {
3     return {
4       x : +x(d),
5       y : +y(d)
6     }
7   })
8 })

```

Once we have defined our model, we can start constructing our chart by:

```

1 var chart = raw.chart();
2 chart.title("Simple scatter plot")
3   .description("A simple scatter plot for learning purposes")

```

To give the user more information of our chart type we should define a title and a description. We could also use an image for the thumbnail by adding it to the `imgs` folder and reference it by `chart.thumbnail(imgs/sample.png)`

Now we want to give the user the opportunity to define additional parameters like width and height for his scatter plot. Therefore, we add:

```

1 var width = chart.number()
2   .title('Width')
3   .defaultValue(900)
4
5 var height = chart.number()
6   .title('Height')
7   .defaultValue(600)
8

```

```

9  var margin = chart.number()
10 .title('Margin')
11 .defaultValue(10)

```

Each of these parameters appears on the left side of the chart and can be changed by the user during runtime. To finalize our simple scatter plot we have to define the `draw()` function. Here we can define what should be drawn and how it should look like using D3. All calculations and styling has to be done inside this function.

```

1  chart.draw(function (selection, data){
2
3  // svg size
4  selection
5  .attr("width", width())
6  .attr("height", height())
7
8  // x and y scale
9  var xScale = d3.scale.linear()
10 .domain([0, d3.max(data, function (d){ return d.x; })])
11 .range([margin(), width()-margin()]);
12
13 var yScale = d3.scale.linear()
14 .domain([0, d3.max(data, function (d){ return d.y; })])
15 .range([height()-margin(), margin()]);
16
17 // let's plot the dots
18 selection.selectAll("circle")
19 .data(data)
20 .enter().append("circle")
21 .attr("cx", function(d) { return xScale(d.x); })
22 .attr("cy", function(d) { return yScale(d.y); })
23 .attr("r", 5)
24 })

```

5.5 Drawbacks of RAW Graphs

RAW Graphs is a very simple tool for visualising data. As we have already shown in our survey about different visualisation tools, the usability of RAW Graphs is amazing. No instructions are needed and the user doesn't have to be a computer science engineer to work with it. But while developing our similarity map we came across some things which can be improved.

5.5.1 Dynamic optional Input Fields

Since we implemented two different algorithms for our similarity map, we needed different input fields in order to let the user choose additional parameters for each algorithm. Currently the user can select one of the two algorithms via a drop-down menu left to the chart. The t-SNE algorithm needs an iteration number, learning rate, perplexity and an early exaggeration as additional parameters, whereas the FDP algorithm only needs the number of iterations. Since RAW Graphs tries to make it as easy to use as possible, it would be a good feature to show these parameters based on a condition, which would be in this situation the type of algorithm.

5.5.2 The draw() Function

A similarity map is a visualisation technique which relies heavily on the parameters and on the number of iterations. So the basic workflow would be to do *trial and error* with these parameters until a good final result is produced. The problem is though, that our algorithms use a random initialization of the coordinates for the data points. This is a common approach and would work very good if there wasn't the `draw()` function of RAW Graphs which is always called after the user changes the parameters. Apart from calling the function on every change to the GUI, RAW Graphs also clears the SVG element in the DOM. This means that for instance after enabling the labels, the whole new scene is redrawn and each data point gets a new random starting position. As a result of the redraw process the whole chart is completely different than before. This makes it very hard to tweak the parameters in order to get good clustering results.

5.5.3 SVG Export

As export formats, RAW Graphs lets the user choose between SVG, PNG and JSON. This is outstanding since RAW is a completely free application. But the SVG format has a big drawback – it is not freely scalable. They use the `width` and `height` attribute which could be completely omitted. Instead of declaring the width and height one could use the `viewBox` feature of SVG which would make the graphic scalable.

Chapter 6

Concluding Remarks

All in all, RAW Graphs is a very useful visualisation tool. It might not be as powerful as other tools out there, but it provides a very straight forward workflow for creating the most common chart types. It produces fast results and is very intuitive to use. This makes it the perfect tool for people who need a more sophisticated visualisation, than for example the common spreadsheet program can provide, but still don't want to spend a lot of time on the layout.

Additionally, RAW Graphs is open-source and may be easily extended. If you know what you are doing, RAW Graphs provides a simple, yet powerful API for defining new chart types. The software is written in a way that implements the *seperation of concerns* principle well. A programmer can therefor extend RAW Graphs by custom charts without the need of understanding much about what RAW Graphs does under the hood. Each chart's logic is contained in a single file which keeps everything clean and concise.

Hence, implementing a similarity map chart was not too complicated. However, we ran into some major problems that come with RAW Graphs. Since a similarity map may be constructed using a variety of algorithms it is desirable to have the option of choosing which algorithm to use for the current data set. Unfortunately RAW Graphs doesn't support dynamic option fields that may only show depending on which type of algorithm is selected by the user. Another problem is the way RAWGraphs exports SVGs. The generated SVG has `width` and `height` properties set which makes the image static and not scaleable. The biggest drawback however without a doubt is the fact that RAW Graphs redraws the whole visualisation from scratch upon any user input. Even if the user is simply changing the color of the visualisation, RAW Graphs will clear the whole SVG created so far and redraw everything from scratch. This might not yield a problem with simple charts such as scatter plots or bar charts, where the result is deterministic and always the same. However, with visualisations that rely on random initializations this becomes a huge problem, since the output will be different on every redraw.

The team behind RAW Graphs states that this behaviour is intended, since RAW Graphs should be used to output static visualisations that may then be used in papers or articles. While this is a valid point, it doesn't help the fact that this design decision leads to poor usability when creating certain charts.

A solution to this, would be to use Principle Component Analysis for the initialization phase of algorithms such as FDP or t-SNE, since PCA is deterministic.

Bibliography

- Abdi, Hervé and Lynne J. Williams [2010]. “Principal Component Analysis”. *WIREs Comput. Stat.* 2.4 [Jul 2010], pages 433–459. ISSN 1939-5108. doi:10.1002/wics.101. <https://doi.org/10.1002/wics.101> (cited on page 10).
- Borg, I. and P.J.F. Groenen [2005]. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005 (cited on page 10).
- Eades, Peter [1984]. “A Heuristic for Graph Drawing”. *Congressus Numerantium 42* [1984], pages 149–160. doi:10.1109/tvcg.2016.2598620. http://www.cs.usyd.edu.au/~peter/old_spring_paper.pdf (cited on page 10).
- Mauri, Michele, Tommaso Elli, Giorgio Caviglia, Giorgio Uboldi and Matteo Azzi [2017]. “RAWGraphs: A Visualisation Platform to Create Open Outputs”. In: *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter*. CHIItaly '17. Cagliari, Italy: ACM, 2017, 28:1–28:5. ISBN 978-1-4503-5237-6. doi:10.1145/3125571.3125585. <http://doi.acm.org/10.1145/3125571.3125585> (cited on page 3).
- RAW Graphs Github* [2018]. <https://github.com/densitydesign/raw/>. Last accessed: 2018-05-13. 2018 (cited on pages 3, 8).
- RAW Graphs Website* [2018]. <https://rawgraphs.io/>. Last accessed: 2018-05-13. 2018 (cited on page 3).
- Van der Maaten, L.J.P. and G.E. Hinton [2008]. “Visualizing High-Dimensional Data Using t-SNE” [2008] (cited on pages 10, 12).