Extending RAWGraphs

A Report on Implementing Custom Chart Types

Elias Doppelreiter, David Egger, Ludwig Reinhardt, Stefan Schnutt

706.057 Information Visualisation 3VU SS 2022
Graz University of Technology

8 July 2022

Abstract

This report gives an overview of how the open-source chart editor RAWGraphs can be extended to include custom chart types. After discussing RAWGraphs in general, the authors write about their experience with their custom extensions. In total, three new chart types were added to RAWGraphs: A paired bar chart, a similarity map, and a chord diagram. In conclusion, the chart implementations using D3 and their integration into the RAWGraphs user interface required a rather significant effort and learning but once the concepts were understood the implementation was straightforward.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

This report aims to give an overview of RAWGraphs [DensityDesign 2022a] from a developer's point of view by extending RAWGraphs with custom chart types. More specifically, the authors extended RAWGraphs Version 2 [DensityDesign 2022b] with a paired bar chart, a similarity map, and a chord diagram.

1.1 Tools and Libraries

The open-source visualization framework D3 [Bostock 2022] is probably the most common framework for interactive visualization on the web. It enables the user to connect data to the Document Object Model (DOM) and subsequently transform the document in the context of the data. D3 focuses on efficient modification of documents that represent data. Large data sets are supported, the framework has excellent performance, and additionally supports animation and other dynamic features. In this project, D3 was primarily used to visualize the user's data and produce a static SVG from it. The offical GitHub repository for D3 boasts 102000 stars, it has 123 contributors and as of the 02 Jul 2022, the latest release was only six days ago. Npm reports almost 1.8 million weekly D3 downloads. The d3 library is distributed under the ISC license.

A GitHub repository was created for this project. The authors development environment was the Ubuntu 22 distribution. Visual Studio Code was used as the IDE. The Discord app was used for communication between the team members. Features like screen sharing and group voice chat as well as a long history of the group chat make Discord a suitable choice.

1.2 Work Preparations

Many documentation files still refer to the original Version 1 of RAWGraphs, which was built using AngularJS. The current Version 2 RAWGraphs was completely rewritten using React [DensityDesign 2022b]. Two repositories are particulary relevant: rawgraphs-chart contains the current implementation of the charts, and rawgraphs-app contains the user interface for RAWGraphs 2.0. Therefore, the authors forked both repositories and started implementing one chart per branch. After all charts were implemented, the authors included the new charts in the user interface and host this new version via GitHub Pages at [Egger et al. 2022] for testing and inspection purposes. The authors plan to open pull requests for all chart branches in order to get their implementations into the official version of RAWGraphs 2.0.

1.3 Chart Code Structure

RAWGraphs requires a specific file structure when a new chart should be added to the project, in essence:

- chartName.js : For exploring purposes.

- index.js : For exploring purposes.
- dimensions.js : This file describes the dimenstions which must be entered by the user in order to succesfully render the specified chart. Dimensions can either be marked as required or optional. Sets of data for one dimension are also possible.
- mapping.js : In this file, the data from the user can be preprocessed if necessary.
- metadata.js: This file describes how the selection of the chart will appear in the RAWGraphs user interface.
- render.js: In this file, all the rendering logic of the chart is located. In the beginning, all visual options are read via object destructuring (at least this was the approach of the authors). Then all necessary chart elements are added via D3 operations.
- visualOptions.js: This file describes the options offered to the user to customize the chart. For example, are color coding, spacing, labeling, and chart-specific parameters.

Chapter 2

RAWGraphs

This Chapter give an overview of the RAWGraphs open-source chart editor. It was taken from the authors previous work on this topic [Doppelreiter et al. 2022].

RAWGraphs was initially created in Milan, Italy, the first Git commit dates back to July 2013 [DensityDesign 2022b]. Responsible for the development and design is the team from the DensityDesign Research Lab at the Politecnico di Milano, consisting of Girogio Caviglia, Michele Mauri, Matteo Azzi, Giorgio Uboldi and Tommaso Elli [Mauri et al. 2017; DensityDesign 2022a; DensityDesign 2022c]. The current software license of RAWGraphs legally binds users to the Apache 2 license.

The vision of RAWGraphs is to provide a connection between vector graphic programs like Adobe Illustrator or Sketch and spreadsheet applications like Excel, Google Sheets or Calc. RAWGraphs can be used to create a number of static chart types. RAWGraphs promises privacy to sensible data if RAWGraph is executed locally on the users PC as well when running RAWGraphs from the public website. The target group of users include designers and visualization geeks hence a variety of partially unconventional chart types are available. Expert users can extend the program and code their own visual representations or charts by using the D3 library and a local installation of RAWGraphs.

2.1 Technical Details

This section covers the technical characteristics of RAWGraphs.

2.1.1 Installation

There are two common ways to get RAWGraphs running. The first and simple way requires the user to open his web browser and navigate to the public website. Building charts from the public website guarantees privacy for all of the users data. One drawback of this method is that the user can't code his own visualizations or charts.

The second way to run the software is by installing RAWGraphs locally, this is for intermediate users. This is done with the help of Git, Node.js, and Yarn. The user clones the official Git repository, installs the required packages, and starts the program with Yarn. Once the service is running, the user may start his browser and open the localhost:3000 port to communicate with RAWGraphs locally. For experts it is also possible to build and upload RAWGraphs to a server.

2.1.2 Input and Output Possibilities

To input the users data, the easiest and probably most popular way, is to drag and drop the users tabular-data file into the user interface. RAWGraphs will parse the values and present a handy preview of the data to the user. Beside the drag-and-drop approach, there are options for pasting the users data, using SPARQL queries, fetching data from a remote URL or importing a proprietary RAWGraphs-project. Data

formats supported are TSV, CSV, DSV or JSON. For now it is not possible to import XML. The authors tested JSON import and it isworking fine. Additionally a wide variety of data samples are provided by RAWGraphs. This sample data actually gets imported from kaggle.

Many options are available to further customize the users input data, this includes adjusting the thousands separator, the decimals separator, the date locale and transform the data to stack by different columns.

RAWGraphs output format include SVG, PNG, JPG as well as the proprietary format .rawgraphs. This enables the users to save their progress and return to it at a later time. SVG graphs are scalable, PNG or JPG are not. It is not possible to create or export animated graphs in RAWGraphs.

2.1.3 Architecture

RAWGraphs is built on top of the D3 library. The user interface is built with React. Other libraries and frameworks that make an appearance are Bootstrap, Bower, Canvas to Blob, CodeMirror, FileSaver.js, is.js, JQuery, NG file upload, Sheet JS and ZeroClipboard. The offical Git repositroy reports that the project consists of 77.3% JavaScript, 13.3% HTML and 9.4% CSS.

2.1.4 Intelligent Features

True intelligent features are not prevalent in RAWGraphs. There are no charts suggested to the user after data insertion. Some charts come out pretty messed up from the automatic initial creation and need manual readjustment to reach an acceptable quality. For example, labels become unreadable, dimensions are inappropriate, or colours are non-existent.

RAWGraphs is not suggesting plot dimensions it rather enforces them. Plot dimensions are scaled automatically which can lead to bad results. For example, when creating a scatter plot (bubble plot) RAWGraphs crops the graphs boundaries to match exactly the extreme mathematical values of the users input data. Sometimes the user does not want this because displaying a specific range is of interest or data points require more room to breath for ground line aesthetics.

There is no adjustment of the software to the user either. RAWGraphs does not remember which chart the user tends to prefer or what kind of settings the user usually takes. It is not possible to log into the application.

2.2 User Interface

The user interface is exceptionally well designed see example figure 2.1. When the authors tested this chart editor, at no point there was the feeling of being lost. The design is intuitive, the use of colours is decent. Only one highly saturated green colour is used to give the user directions and help him navigate without thinking. The user is taken by the hand when creating a chart. The whole process is split up in five steps and only after the user finishes his momentarily step, the succeeding step is revealed. This automatically focuses the user attention to where it should be.

At Step three, a crucial step where the user has a lot of options which can lead to errors, RAWGraps gives immediate feedback to what and where the problem is. Tool-tips provide additional information to icons. When customizing the appearance in step four, values that are not adjustable are greyed out. The user can select his own custom colours via a colour picker popup. Unfortunately there is no pipette or the ability to save the users palettes.

It is definitely worth mentioning that the documentation of RAWGraphs is solid. Most of our questions of different depth could be answered with ease. The installation manual is short and concise. For almost every diagram there is a link from the user-interface to a Youtube video that showcases this diagrams creation process, as well another link that points to the contextual code on the GIT-repository. The API is well documented and even a scientific paper is shared that gives insight at an academic level.

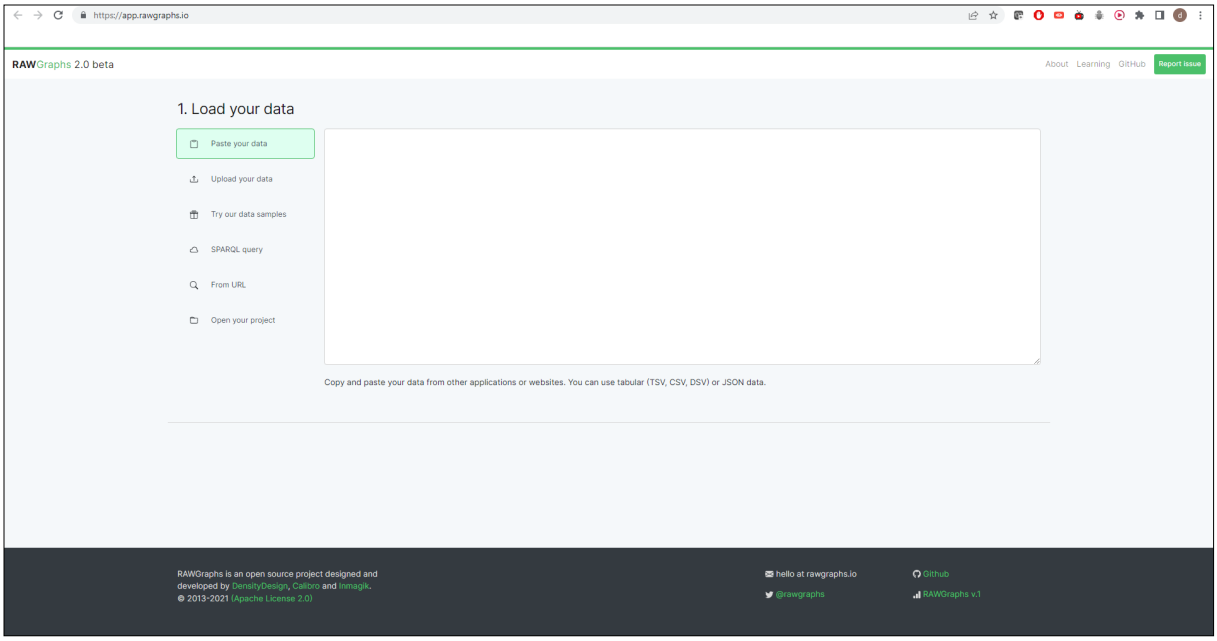

Figure 2.1: The landing page of RAWGraphs. [Screenshot taken by the authors of this report.]

2.3 Advantages and Disadvantages

Advantages are the UX, privacy for both the official website, wide variety of graphs including non conventional, good volume of input and output data formats and that the software is also interesting for power users. Disadvantages are that installing RAWGraphs locally requires intermediate computational skills and that the charts are not fully customizable for example it's not possible for the user to define a custom range for the axis of charts, although it's possible to include zero. Another problem, especially for power users, is that there is no feedback on the maximum amount of data that can be processed by RAWGraphs. If the user happens to exceed the systems or browsers limitations RAWGraphs will just crash. One thing the authors would love to see is an undo function where the user can step back previous inputs in case of mistakes.

2.4 Conclusion

RAWGraphs has some years of love and experience behind its back and the authors can certainly feel and see that. At one point, RAWGraphs got crowdfunded to fuel further development. Although not perfect, when compared to other open source chart editors, RAWGraphs earned its place among the top ranks featuring exquisite user experience, an above average number of chart types, privacy, and options to serve a broader spectrum of users from casual users to visualization veterans.

Chapter 3

Paired Bar Chart

Paired bar charts, or pyramid charts, are a common way of comparing two populations which depend on the same variable. A common example is an age population pyramid, showing the number of men and women from a particular population in various age bands.

3.1 Approach

The author started by having a good look at both RAWGraphs repositories rawgraphs-app and rawgraphs-chart. In the latter one, documentation for adding new charts was found. By following this step-by-step instruction documentation, a first test chart, a scatterplot, could be added to the rawgraphs-chart repository. At first, the implementation could only be seen and tested inside the rawgraphs-charts repository by executing a sandbox for showing the chart render results from specific test data. After completing this test chart, the author realized that fundamental knowledge about D3 was necessary to implement the paired bar chart. Therefore, the author followed some tutorials about how to create basic D3 charts. The most important learned things were the data/join functionality of D3, selections, scaling, creating axes as well as how to keep D3 code organized.

After understanding D3 sufficiently well, the real implementation of the paired bar chart began. As there were also other types of bar charts in the implementation some functionality like the ordering function could be reused. When the chart looked good in the sandbox execution the author began by including the chart in rawgraphs-app and testing the customization options of the user. The next step was to fine-tune these options. Finally, an appropriate sample data set was chosen and included in the application, which finished the chart implementation.

3.2 Sample Dataset

The data for testing the paired bar chart was taken from StA [2022]. The goal was to get a chart from the population of Austria with bars on the left side indicating how many women are at a certain age while the bars on the right side show how many men there are in the same age group. To get a useful visualization it was decided to prepare the data and group people in 5-year intervals with one bar representing such an interval. The resulting paired bar chart can be seen in Figure 3.1.

3.3 Implementation

The rendering process of the chart begins with loading all visual options and data from the user. The next step is the execution of the following functions, which are all included in the render function:

- calcProps(): Here, all additional properties which are needed for creating the chart like spacing, ordering, and accessing the data are calculated.

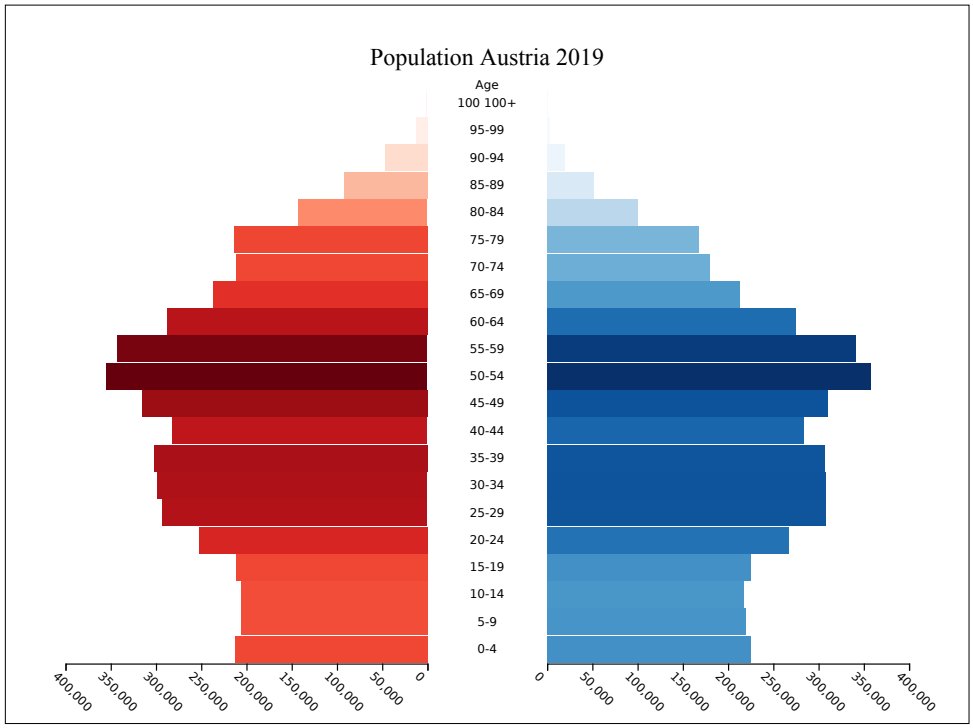

Figure 3.1: Paired Bar Chart: An example paired bar chart with female (left) and male (right) populations in Austria in 2019. The data is from StA [2022].

- createBounds(): This function returns the group element inside the SVG element without the margin edges, or in other words the element in which the chart will be rendered.
- createScales(): This function creates and returns the scales which are needed to correctly render the axes and bars of the chart. The two horizontal variables both have a linear scale. The left one needs a reverse scale, since it lists the data in the other direction. For the vertical variable (y-axis), a band scale is used to pick out each category of the data and assign them all an equal amount of space.
- createAxes(): Depending on the previous calculation of the scales, this function adds the axes of the three variables to the chart.
- createAxisLabels(): Depending on the user's choice, the axis labels of the chart are shown and customized here.
- createBars(): In this method, the bars are added to the chart for both sides, depending on the previously calculated scales.

3.4 User Interface

When creating a paired bar chart via RAWGraphs the user first uploads the data to be in the chart, or uses the sample dataset of the population of Austria in 2019 [StA 2022], which is included in the application. The user then selects the kind of chart, as shown in Figure 3.2.

After the chart is selected the user must specify the data columns to be included in the chart via drag and drop. Three variables are required to render the chart. These are numerical variables for the left and right side of the chart, as well as the common variable which is displayed vertically (Y Axis) and can be of any type (date, numeric, text). This can be seen in Figure 3.3.

After assigning all three required variables, the user is presented with the chart visualization. It is now

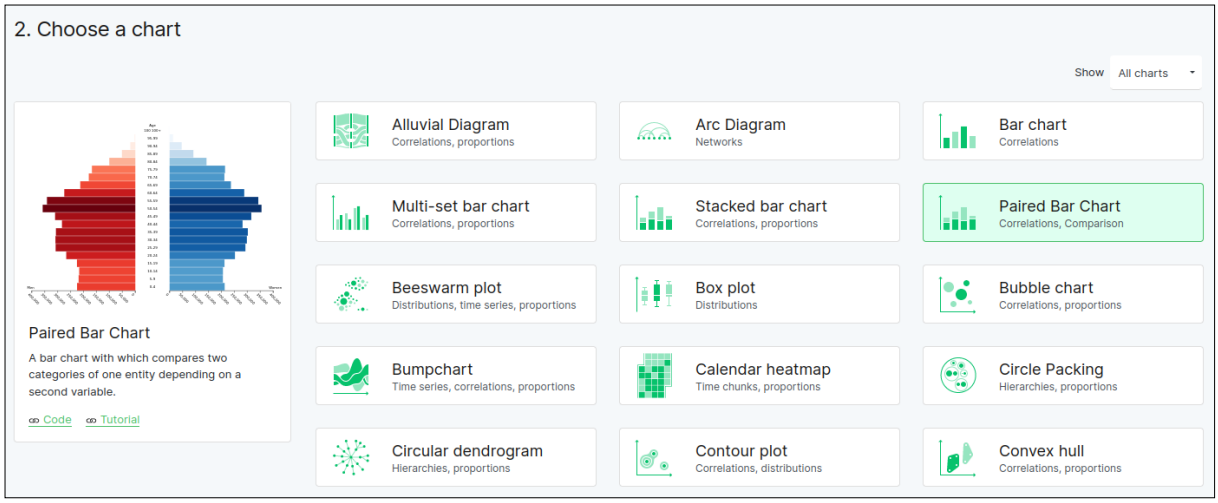

Figure 3.2: Paired Bar Chart: Selection of a Paired Bar Chart in the RAWGraphs application.

Figure 3.3: Paired Bar Chart: Mapping the correct dimensions to the three variables (Left Side, Right Side, and Y Axis) for a paired bar chart.

possible to adjust the chart in the following ways:

- Artboard: The parameters in this section define the basic spacing of the chart as well as the background color and the (optional) chart title (see Figure 3.4).
- Axis: All properties of the axes which can be customized can be changed in this section. This includes the (optional) labeling of the axes and the alignment and rotation of the tick labels (see Figure 3.5).
- Chart: Here the user can specify the particular parameters of the paired bar chart. These are the spacing between the two opposing bar charts, the vertical order of bars, and the padding of the bars (see Figure 3.6).
- Color: In this section, it is possible to customize the color of the two opposing bar charts (see Figure 3.7).

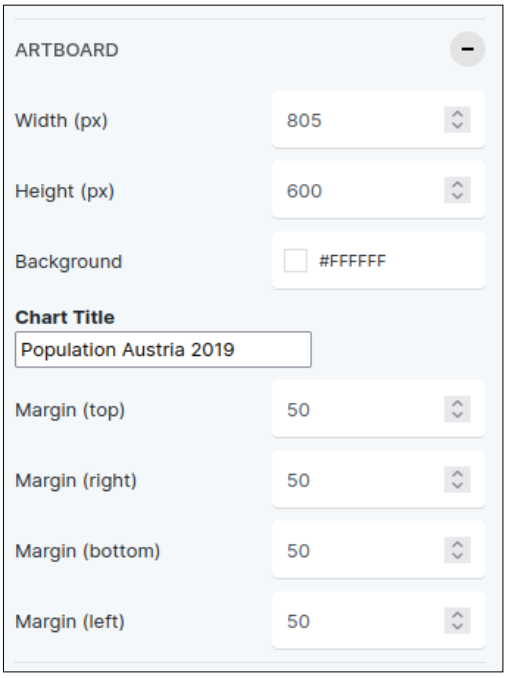

Figure 3.4: Paired Bar Chart: Artboard options.

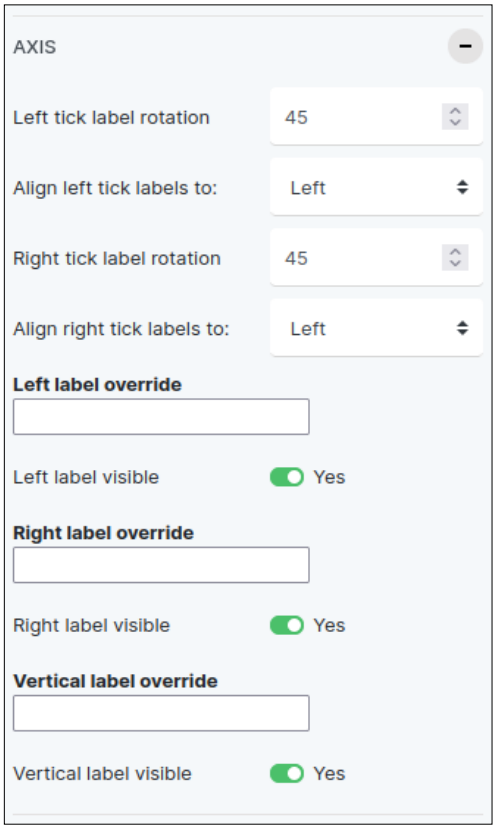

Figure 3.5: Paired Bar Chart: Axis options.

Figure 3.6: Paired Bar Chart: Chart-specific options.

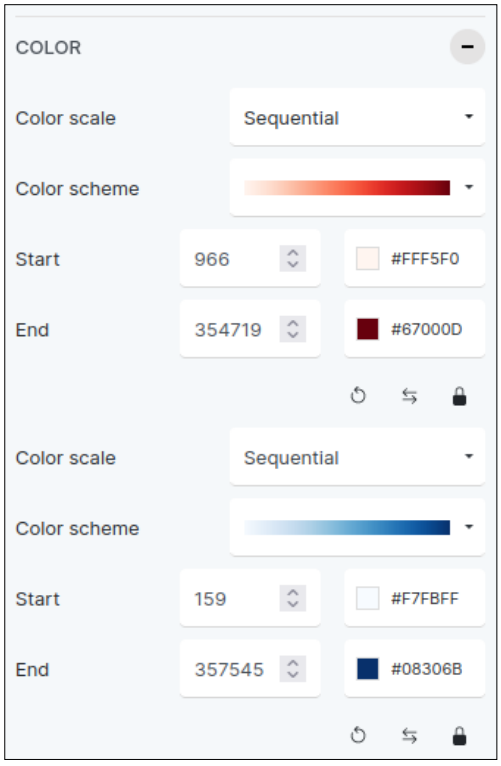

Figure 3.7: Paired Bar Chart: Color options.

Chapter 4

Similarity Map

A similarity map takes data points in a high-dimensional space and plots them in two dimensions, preserving the distances between the points as far as possible. Points close together (similar points) in the original, high-dimensional space should be close together in the resulting similarity map. The technique is often refered to as dimensionality reduction. RAWGraphs charts use two-dimensional Cartesian coordinate systems to visualize their data. Hence, the authors similarity map chart also projects the data points onto two dimensions.

4.1 Tools

tSNEJS is a JavaScript library [Karpathy 2016] of the t-SNE dimensionality reduction algorithm [van der Maaten and Hinton 2008; Wattenberg et al. 2016]. This library is simple and efficient with no superfluous features. The reason why this library was chosen by the authors is that this library was returned in the top results of a Google search with the keyword "tsne js" and that the library is efficiently documented, featuring a simple and to the point manual as well as an interactive web demo. The web demo allowed the developer to interactively tinker around, which greatly aided grasping the new technology. The tsnejs library is distributed under the MIT license.

DruidJS is dimensionality reduction library with great performance and a wide variety of dimension reduction algorithms [Cutura et al. 2020]. Initially, the plan was to include this sophisticated tool into the project, but due to problems in the development phase this plan was discarded in order to make progress and meet deadlines.

4.2 Approach

The developer started by getting in touch with the D3 functionality. The data join concept and the syntax of D3 programs was explored. After experimenting with simple graphs a scatter plot was created. This was the foundation for the similarity map chart. Initially, the developer expected that the similarity had to be calculated individually for every cell of the data set. It turned out that the dimensionality reduction algorithms calculates similarity internally. Selecting which data is used for calculation is ultimately a decision the user has to make inside RAWGraphs, where it will be directly communicated which type of data is expected for the computation via the user interface.

At first, the DruidJS library was included to serve as the dimensionality reduction algorithm solution. The concept of the DruidJS library and its promises were perfect. Different dimensional reduction algorithms in one package including high performance and some extra features like deterministic calculation. But due to the unconventional documentation of the DruidJS library, where practical examples are based inside an observablehq environment, it was discarded for the moment. The observablehq environment made it unnecessarily difficult to tinker with and to perform basic operations. It came to a point where

the developer made the decision to try out other software solutions for dimensionality reduction, in honor of focusing on progress rather than perfection. Hence, the developer decided to try out different t-SNE libraries. Already the next library which was the tsnejs library has proven to be practical and work was continued towards a fully functional t-SNE similarity map implementation.

Fusing the scatter plot foundation and the tsnejs library was relatively fast. At this point, the developer wanted to get the functioning dimensional reduction code into the RAWGraphs repository and this is where some serious trouble started. For undiscovered reasons it was impossible for the developer to get the official RAWGraphs repository running on the developers Ubuntu 22 distribution as well as the developers Windows 7 partition. Numerous hours were invested into trying to get the framework to work which resulted in a workaround.

Carefully following the official instructions just did not cut it. It ended up in a very frustrating experience where a lot of exotic measures were taken to find a solution. At some point, it felt that with every step that was successfully completed to solve one error message a new one emerged. For some of the error messages, there was even a lack of information on Stack Overflow or Google.

This was starting to become a serious problem as the deadlines were closing in fast. There was a locally running t-SNE chart available and the sandbox for the RAWGraphs repository was luckily functioning, this allowed the developer to code the visualization, but not the interaction with the user interface. Continuous pressure was ramping up and at one point the developer was not sure to be able to deliver any working solution at all. The other teammate which was already further ahead with the paired bar chart was already at capacity and now this other teammate had to take over the similarity map developers part regarding the RAWGraphs side. It is worth mentioning that the RAWGraphs repository was reportedly running for two of the four team members, where one team member never tried to locally start the RAWGraphs app.

Finally, with combined efforts the t-SNE implementation was up and running. Without the technical problems encountered a considerable amount of time could have been allocated towards improvements. There was no time to include additional dimensional reduction algorithms like the PCA algorithm. The requested feature where the user can switch between stochastic or deterministic calculation of the t-SNE graph is not possible to implemented at this point. This feature is not supported in the currently implemented tsnejs library. Since dimensional reduction algorithms create scales that have no real relevance, the axis labels are disabled by default.

4.3 Implementation

4.3.1 Data Preprocessing

The whole data preprocessing is done inside the RAWGraphs engine. This allows quite flexible input types like TSV, CSV, DSV or JSON data. Additionally, the user can fetch data from URL, run a SPARQL query, or use the existing data samples which are primarily from Kaggle. After RAWGraphs made sure that the user did not provide unreasonable input, the data is passed to the similarity map.

4.3.2 Program Structure

The program flow is shown in Figure 4.1. The functions involved are:

- calcProps(): First, the chart proportions are calculated, including the physical scales of the total chart, the D3 x- and y-accessors based on the input data, and the call of the t-SNE algorithm.
- calcReducedDimensions(): The t-SNE algorithm from the tsnejs library performs the dimensionalality reduction, calculating the new projected 2d coordinates for each data point.
- createBounds(): To create the chart bounds the SVG that already exists inside the DOM is manipulated using the D3 functionality. First, a rect is appended to the SVG, followed by adding the title text. The last step appends a group to the SVG which is used to transform the charts boundaries.

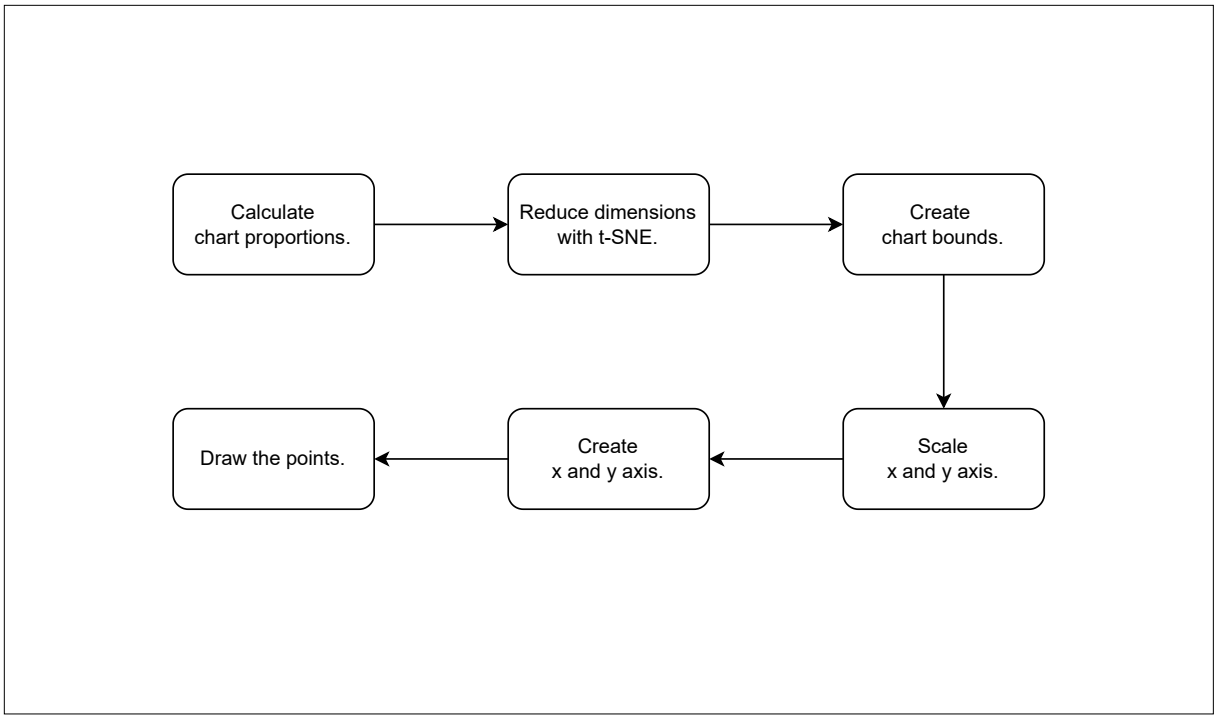

Figure 4.1: Similarity Map: Program flow.

- createScales(): This is done explicitly with the respective D3 functions. The x-axis and y-axis are scaled linearly in relation to the previously calculated dimensions and boundaries. The D3 nice function makes sure that each axes are enclosed by rounded extreme values which gives the chart a professional polish.
- createAxes(): In order to create the x-axis and y-axis the D3 fucntions axisLeft and axisRight are called. The previously calculated scales are applied. The format of the ticks is modified to make the axis caption more readable. Additionally, the axes are transformed to represent an appropriate scale.
- drawScatterPoints(): Finally, the indiviual data points are plotted to the SVG. This is done with the famous D3 data join feature. Circle elements are created with a predefined radius. The coordinates are calculated based on the previously computed scales parametrized by the x- and y-accessors.

4.3.3 Dataset

For demonstration purposes the Iris dataset was chosen. This is a standard RAWGraphs dataset. It is the famous database from Fisher 1936, which is used frequently in machine learning examples. The dataset consists of 150 entries describing three different flower types. Only one class of the three is linearly separable. This makes it a great dataset for testing a dimensionality reduction algorithm.

4.3.4 Dimensionality Reduction

Listing 4.1 shows how the tsnejs library is used. The tsnejs library recieves two parameters defined by the user called epsilon and perplexity. Epsilon is the learning rate which decides the step size of a gradient descent. By setting epsilon too small the algorithm's run time increases exponentially. On the other hand, an epsilon that is too large can cause to miss the minimum of the gradient which may result in distances between the data points that are too large. Perplexity describes the number of randomly chosen data points at a time which are used to identify local structures. It is recommended to scale perplexity with the size of the data. The number of dimensions for the projection is always set to two, since RAWGraphs exclusively features 2D charts.

```
function calcReducedDimensions() {
    const opt = {dim : 2, epsilon, perplexity}
    var tsne =  allTSNEE(opt)
    const tsneData = data.map(row => {
      return row.dimensions
    })

    tsne.initDataRaw(tsneData)

    for(var k = 0; k < 500; k++) {
      tsne.step();
    }

    return tsne.getSolution();
  }
```

Listing 4.1: Similarity Map: Dimensionality reduction with t-SNE.

Calling the initDataRaw function with the user's data as the parameter initializes the t-SNE algorithm accordingly. Now that the parameters are set and the data is initialized, t-SNE is ready to run which is done iterative in 500 cycles. Every time the tsne.step() function is called the solution gets clearer until a threshold is reached. Finally, calling the tsne.getSolution() function returns an array of 2d points. The 2d coordinates are passed forward to continue with the visualization in the chart creation pipeline.

4.4 User Interface

Creating a similarity map requires four steps. In Step One, the user selects an appropriate dataset. The authors recommend the Iris dataset for demonstration purpose. Step Two is depicted in Figure 4.2, in which the user selects the similarity map as the chart type. The similarity map can be found in the second column, the fourth from below. Step Three is mapping the data to the variables, as shown in Figure 4.3. Finally, in Step Four, before exporting the finished chart as SVG, the user can optionally customize the similarity map, as shown in Figure 4.4. After the user is done with the customization, the final result can be downloaded as an SVG and might look like the image in Figure 4.5.

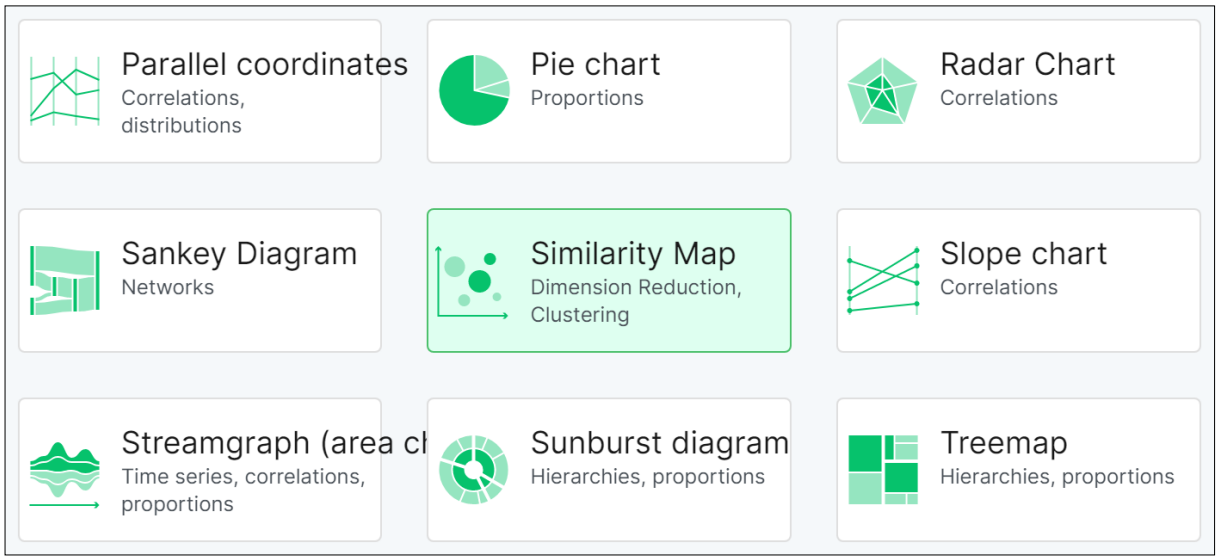

Figure 4.2: Similarity Map: Step 2, choosing a chart. [Screenshot taken by the authors of this report.]

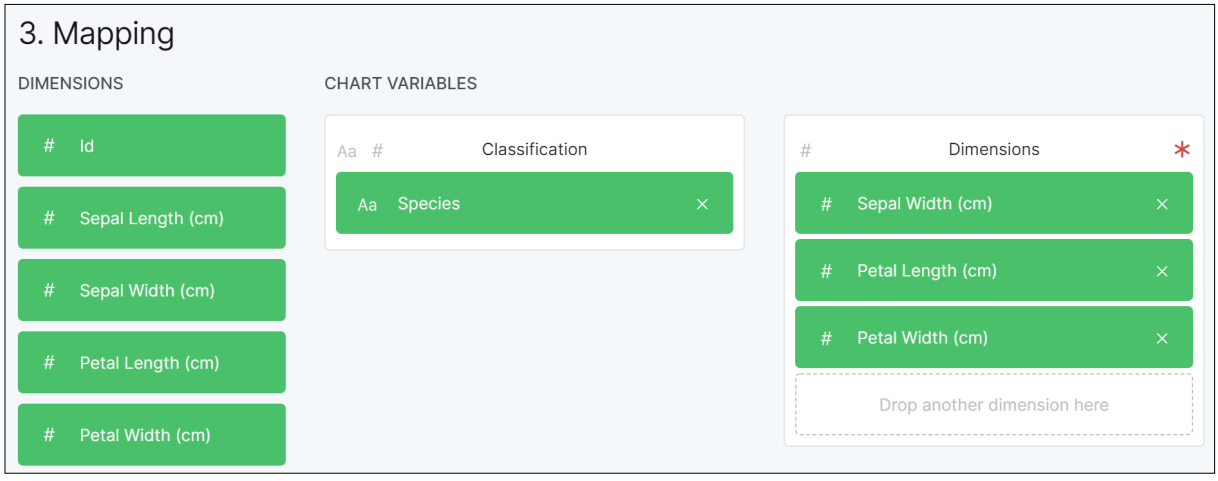

Figure 4.3: Similarity Map: Step 3, mapping data to variables. [Screenshot taken by the authors of this report.]

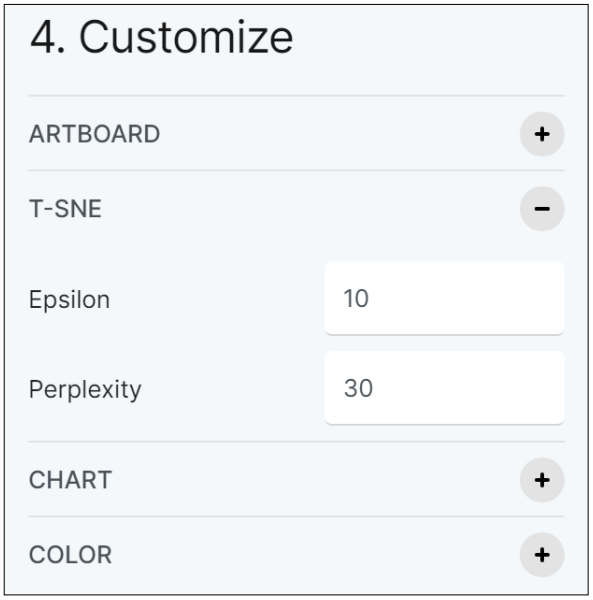

Figure 4.4: Similarity Map: Step 4, customization. [Screenshot taken by the authors of this report.]

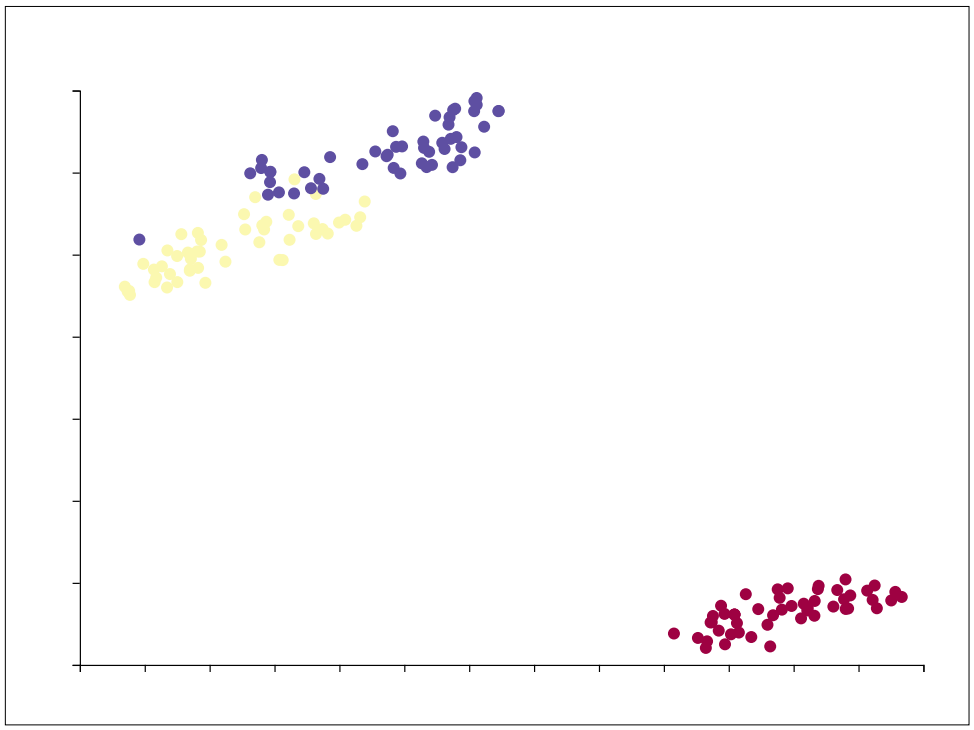

Figure 4.5: Similarity Map: Final similarity map as SVG. [SVG created by the authors of this report.]

Chapter 5

Chord Diagram

Chord diagrams are a special type of chart, which can be used to represent directed graphs, more specifically the relationships between individual graph nodes. Conceptually, this makes chord diagrams similar to arc diagrams. Since RAWGraphs already features arc diagrams, we used the existing arc diagram implementation as a starting point for our chord diagram chart.

5.1 Sample Dataset

For a sample dataset, we chose a migration flow dataset of migration flows between 196 countries in 5-year intervals in the period from 1990 to 2010 from Abel and Sander [2014]. The regions or countries act as nodes which are connected by directed edges with a given magnitude, i.e. the number of people who migrated from one region or country to another. The dataset was added to RAWGraphs sample datasets. The concrete data file in CSV format was downloaded from a public website hosted by the authors [Sander et al. 2022] and used under Creative Commons for our project. A small portion of the data is shown in Figure 5.1.

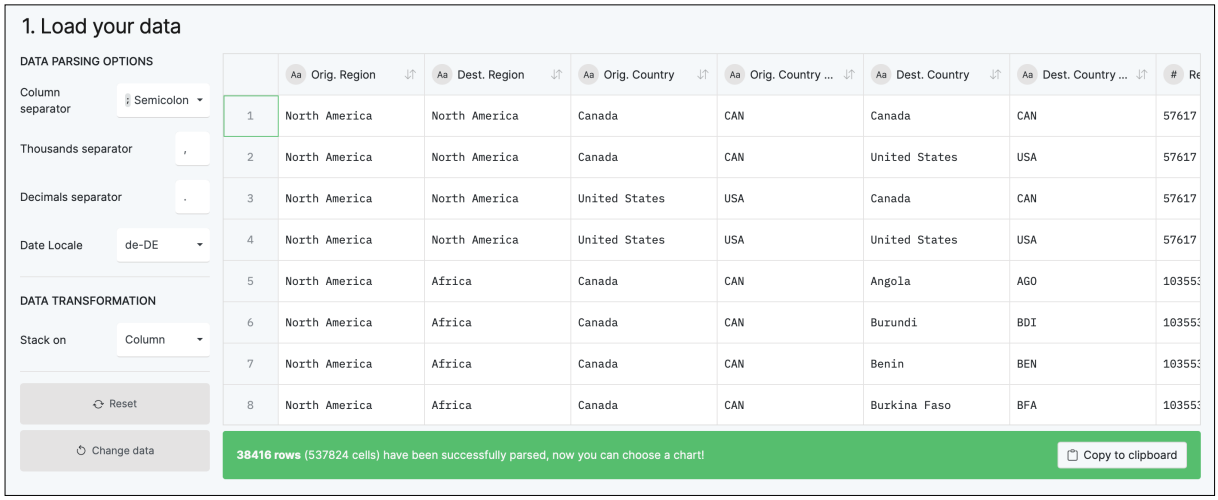

Figure 5.1: Chord Diagram: Migration flow dataset. [Screenshot was taken by the authors of this paper.]

5.2 Implementation

The chord diagram was created in D3 based on the example implementations by Mike Bostock [Bostock 2018a; Bostock 2018b]. To render the chord diagram, following helper functions were implemented:

- matrixFromData(): Users need to specify an origin column and a destination column. However, for the further implementation, an adjacency matrix representation of the datais used with the connection strengths as values.
- createChords(): The D3 library provides a method to create chords using a data matrix. The createChord function wraps the D3 implementation, configures chart options, and calls the D3 chords function.
- drawRing(): In this function, the outer ring of the chord diagram is created and added to the SVG element, depending on the specified inner and outer radius which are used for the ring thickness.
- drawChords(): Analogous to the rings, the inner chords connecting the categories or groups are created and added to the SVG element. Here, the diction of the chords is specified as well as the different radii used to distinguish source and destination chords more easily.
- drawGroupLabels(): In this last helper function, the group labels are created and added to the SVG element. The group's start and end angle are used to calculate the middle angle to position the label in the center of groups. Further, this middle angle is then used to rotate the labels around the chart settings.

To integrate the D3 chart into RAWGraphs, we used the RAWGraphs implementation of the arc diagram [mikima 2021] as the basis for the port. In summary, the chart was created in the render function provided by RAWGraphs, which passes the SVG element as a parameter. Using this SVG element and the above helper functions, the chord diagrams is rendered.

5.3 User Interface

The user interface of the chord chart is very similar to the user interface of most other chart types that can be created using RAWGraphs 2.0. The main difference between the chord diagram and the other chart types is the mapping of the data and the configuration options available. The following explains how to create a chord diagram using our chord diagram implementation for RAWGraphs 2.0:

1. First of all, it is necessary to select the data for visualization. For this, we included a sample migration flow dataset into the RAWGraphs app. The dataset is described in more detail in Section 5.1. In the RAWGraphs app, the migration flow dataset can be selected under *Try our data samples* (see Figure 5.2).
2. In the second step, it is necessary to select the chord diagram from the different chart types available in RAWGraphs.
3. In the third step, it is necessary to map the dimensions of the dataset to chart variables. To create a chord diagram, it is necessary to map the source node, the target node and the size. Size describes the flow from the source node to the target node (see Figure 5.3).
4. In the final step of creating a chord diagram, various configurations can be made to the chord diagram. Depending on the number of paths in the dataset, it may be necessary to manually adjust the size of the diagram. In addition to the configuration option for the size of the diagram, there are several other configuration options available, including the configuration option for colors, margins, paddings, and labels. Figure 5.4 shows some of the configuration possibilities and the resulting chord diagram.

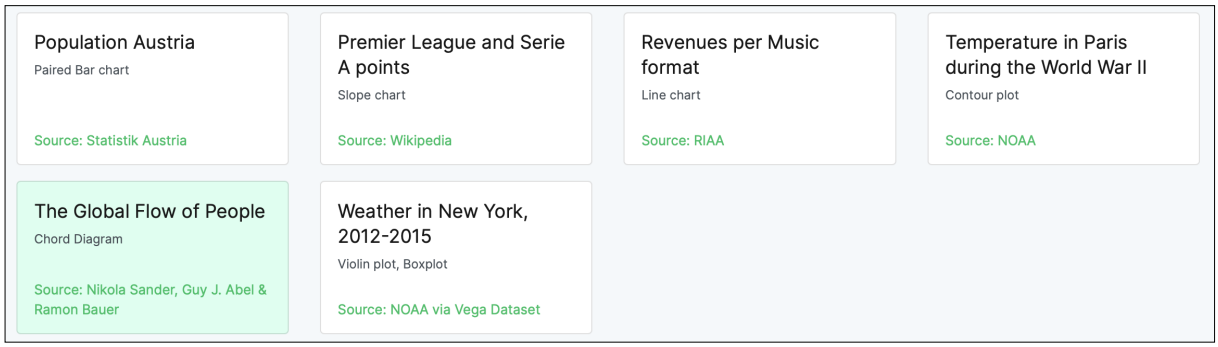

Figure 5.2: Chord Diagram: Selection of the migration flow dataset. [Screenshot taken by the authors of this paper.]

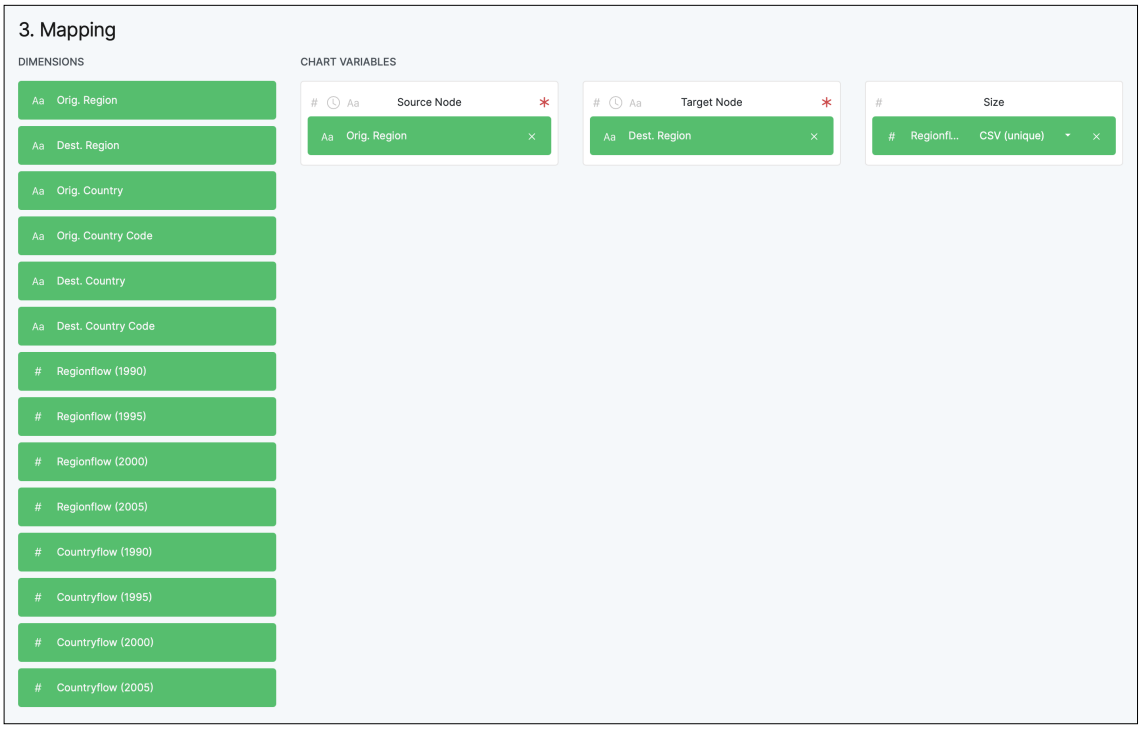

Figure 5.3: Chord Diagram: Mapping of the dataset. [Screenshot taken by the authors of this paper.]

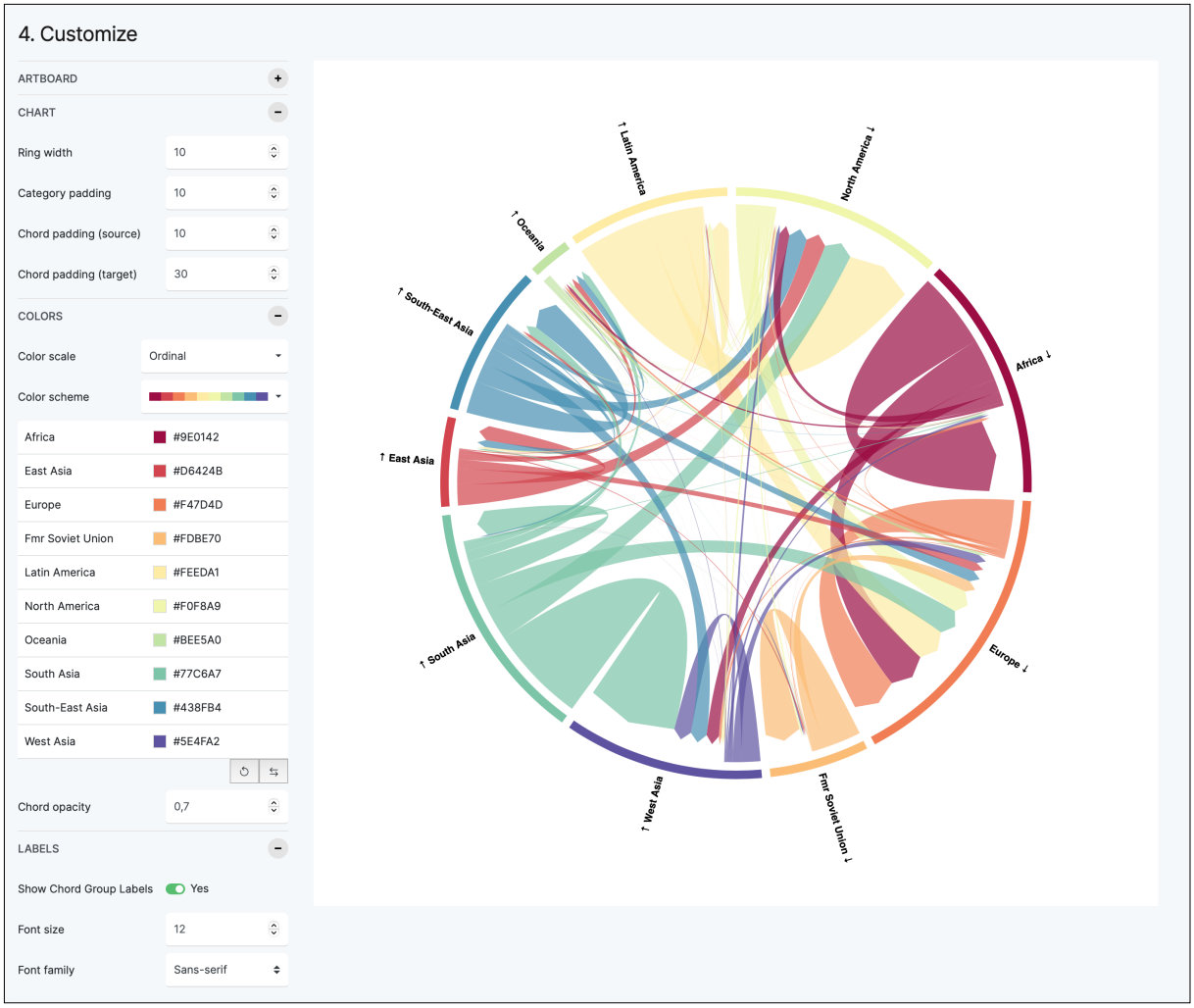

Figure 5.4: Chord Diagram: Visual options and resulting diagram. [Screenshot taken by the authors of this paper.]

Chapter 6

Concluding Remarks

Chart editors such as RAWGraphs are essential information visualization tools used to present, and more easily understand and analyze information and data. RAWGraphs is a great open-source chart editor, which allows users to create numerous chart types. Thanks to its self-explanatory user interface, we see great potential for the tool for the general masses.

To implement a custom chart type for RAWGraphs, it is necessary to adhere to a strict custom structure. Since the RAWGraphs documentation is lacking in many points, following their online example did not work out as expected. Therefore, a lot effort went into debugging and learning the RAWGraphs specifics. Further, most charts in RAWGraphs are implemented using the D3 library. Since none of the authors had experience with D3, there was a learning effort as well. However, once the concepts were understood, the actual implementation of the charts was straightforward. There were slight variations in the data mappings, but the overall code except for the chart rendering was very similar.

During the project, the authors were in contact with the RAWGraphs team. Since the team was interested in integrating our chart implementations developed during this course, there is future work left. Among other steps, this might include refining the code and creating a merge request on the official RAWGraphs Git repositories hosted on GitHub. Seeing our changes integrated into a real project is a neat reward for the efforts.

Bibliography

Abel, Guy J. and Nikola Sander [2014]. *Quantifying Global International Migration Flows*. Science 343.6178 (28 Mar 2014), pages 1520–1522. doi:10.1126/science.1248676 (cited on page 19).

Bostock, Mike [2018a]. *Chord Dependency Diagram*. 06 Oct 2018. https://observablehq.com/@d3/chord-dependency-diagram (cited on page 20).

Bostock, Mike [2018b]. *Chord Diagram*. 06 Oct 2018. https://observablehq.com/@d3/chord-diagram (cited on page 20).

Bostock, Mike [2022]. *D3.js - Data Driven Documents*. 07 Jul 2022. https://d3js.org/ (cited on page 1).

Cutura, Rene, Christoph Kralj, and Michael Sedlmair [2020]. *DRUIDJS — A JavaScript Library for Dimensionality Reduction*. Proc. IEEE Visualization Conference (Vis 2020) (Online). 25 Oct 2020, pages 111–115. doi:10.1109/VIS47514.2020.00029. https://renecutura.eu/pdfs/Druid.pdf (cited on page 13).

DensityDesign [2022a]. *RAWGraphs*. 15 May 2022. https://rawgraphs.io/ (cited on pages 1, 3).

DensityDesign [2022b]. *RAWGraphs GitHub Repository*. 15 May 2022. https://github.com/rawgraphs/rawgraphs-app (cited on pages 1, 3).

DensityDesign [2022c]. *RAWGraphs Online Documentation*. 15 May 2022. https://rawgraphs.io/rawgraphs-core/docs/api/ (cited on page 3).

Doppelreiter, Elias, David Egger, Ludwig Reinhardt, and Stefan Schnutt [2022]. *Open-Source Chart Editors*. 706.057 Information Visualisation SS 2022 Survey Paper. Graz University of Technology, 24 May 2022. https://courses.isds.tugraz.at/ivis/surveys/ss2022/ivis-ss2022-g1-survey-chart-editors.pdf (cited on page 3).

Egger, David, Elias Doppelreiter, Ludwig Reinhardt, and Stefan Schnutt [2022]. *RAWGraphs Custom Version*. 07 Jul 2022. https://blindguardian50.github.io/rawgraphs-app/ (cited on page 1).

Karpathy, Andrej [2016]. *tSNEJS*. 24 Oct 2016. https://github.com/karpathy/tsnejs (cited on page 13).

Mauri, Michele, Tommaso Elli, Giorgio Caviglia, Giorgio Uboldi, and Matteo Azzi [2017]. *RAWGraphs: A Visualisation Platform to Create Open Outputs*. Proc. 12th Biannual Conference of Italian SIGCHI Chapter (CHItaly 2017) (Cagliari, Italy). ACM, 18 Sep 2017, 28:1–28:5. ISBN 1450352375. doi:10.1145/3125571.3125585 (cited on page 3).

mikima [2021]. *RAWGraphs Charts Repository*. 18 Oct 2021. https://github.com/rawgraphs/rawgraphs-charts/tree/master/src/arcdiagram (cited on page 20).

Sander, Nikola, Guy J. Abel, and Ramon Bauer [2022]. *The Global Flow of People*. 03 Jul 2022. http://www.global-migration.info/ (cited on page 19).

StA [2022]. *Austrian Population 2019*. Statistik Austria, 04 Jul 2022. https://statistik.at/statistiken/bevoelkerung-und-soziales/bevoelkerung/bevoelkerungsstand/bevoelkerung-nach-alter/geschlecht (cited on pages 7–8).

Van der Maaten, Laurens and Geoffrey E. Hinton [2008]. *Visualizing High-Dimensional Data Using t-SNE*. Journal of Machine Learning Research 9 (2008), pages 2579–2605. https://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf (cited on page 13).

Wattenberg, Martin, Fernanda Viégas, and Ian Johnson [2016]. *How to Use t-SNE Effectively*. Distill (13 Oct 2016). doi:10.23915/distill.00002. https://distill.pub/2016/misread-tsne/ (cited on page 13).