TU WIEN Informatics

Interactive Visual Exploration of Large Bipartite Graphs using Firework Plots

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Visual Computing

eingereicht von

Katharina Unger
Matrikelnummer 01325652

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Univ.Ass. Dr.techn. Manuela Waldner, MSc

Wien, 30. April 2020

Katharina Unger	Eduard Gröller

TU WIEN Informatics

Interactive Visual Exploration of Large Bipartite Graphs using Firework Plots

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Visual Computing

by

Katharina Unger
Registration Number 01325652

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Univ.Ass. Dr.techn. Manuela Waldner, MSc

Vienna, 30th April, 2020

Katharina Unger	Eduard Gröller

Erklärung zur Verfassung der Arbeit

Katharina Unger

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. April 2020

Katharina Unger

Acknowledgements

First of all, I would like to thank my advisors Manuela Waldner and Eduard Gröller for their guidance and feedback during the thesis. Moreover, I want to thank them that they offered me a place in their office where I could work without any disturbance.

I also want to thank my boyfriend, Markus, for his emotional support during my thesis.

Last but not least, I would like to thank my family for supporting me during my studies and allowing me to concentrate on my studies without worries.

Kurzfassung

In dieser Arbeit präsentieren wir eine webbasierte interaktive Explorations-Schnittstelle für die breite Masse, die das Untersuchen von großen, gewichteten, bipartiten Graphen ermöglicht. Unsere Hauptmotivation ist die österreichische Medientransparenz-Datenbank, welche aufgrund eines Gesetzes von der österreichischen Bundesregierung erstellt wurde. Das Gesetz schreibt Rechtsträgern vor Werbe- und Subventionszahlungen an Medienunternehmen zu veröffentlichen.

Es gibt bereits viele interaktive Explorationswerkzeuge, jedoch verwenden die meisten oft sehr komplexe Visualisierungen, die für Experten aus den entsprechenden Domänen gedacht sind. Da es jedoch auch viele Datensätze gibt, die für ein großes Publikum von Interesse sind, wie zum Beispiel die Medientransparenz-Datenbank, stellen wir ein Rahmenwerk zur Verfügung, welches nicht nur für Fachleute, sondern auch für Laien gedacht ist.

Daher haben wir eine Web-Applikation entwickelt, die für die breite Masse zugänglich ist. Außerdem haben wir systematisch den State-of-the-Art von webbasierten Rendering-Technologien untersucht, welche uns die aktuellen Limitierungen dieser Technologien aufgezeigt haben. Darüber hinaus haben wir auch die Leistung von verschiedenen Bibliotheken analysiert, um die effizienteste Lösung für unsere Implementierung zu finden.

Wir stellen unser Konzept der Firework Plots vor, welches eine gut skalierbare, gewohnte Darstellung ermöglicht. Dieses Konzept basiert auf Node-Link Diagrammen in Kombination mit unterschiedlichen Visualisierungs- und Interaktions-Konzepten. Wir verwenden hierarchische Aggregation, um Skalierbarkeit sicherzustellen. Außerdem verwenden wir kraftbasierte Graphenanordnungen, welche schichtenweise mithilfe von Restriktionen berechnet werden, um das Verfolgen von Knoten über die unterschiedlichen Hierarchiestufen zu ermöglichen. Dieser Prozess wird durch Animationen und stufenloses Zooming unterstützt. Darüber hinaus haben wir auch eine effiziente Verwaltung von sichtbaren Geometrien implementiert, um visuelle Uberladung zu reduzieren und das Laufzeitverhalten zu verbessern.

Basierend auf den Analysen der webbasierten Rendering-Technologien haben wir ein Rahmenwerk und die Firework Plots implementiert. Wir zeigen die Brauchbarkeit unserer Implementierung mithilfe verschiedener Anwendungsfälle und vergleichen sie auch mit einem verwandten Ansatz. Außerdem haben wir mehrere Analysen durchgeführt, um die Effizienz unseres Programs zu zeigen

Abstract

In this thesis, we introduce a web-based interactive exploration interface for a broad audience to investigate large, weighted, bipartite graphs. The motivation of this work is based on the *Media Transparency Database* which arises from an Austrian law that compels legal entities to announce their advertisement spendings to media organizations and meets the specified characteristics.

Most current interactive exploration tools use complex visualizations because they were developed for domain experts. As the Media Transparency Database is of potential interest to a broad audience, we provide a framework not just for domain experts but also for inexperienced users.

Therefore, we conducted systematic benchmarks to compare state-of-the-art web-based rendering techniques. Furthermore, we compared the performance of different libraries to determine the most efficient rendering solution and current limitations of web-based rendering.

We introduce the concept of Firework Plots, which aims to provide a common visualization that scales well with the size of the data. Our visualization concept is based on intuitive node-link visualization in combination with multiple visualization and interaction concepts. Hierarchical aggregation is used to improve scalability. Constrained, layered, force-based graph layouts, as well as firework animations and seamless zoom, are used to allow inexperienced users to drill down the graph hierarchy and track nodes through the hierarchy. Moreover, visibility management is used to reduce clutter and improve performance.

Based on the insights of our web-based graph rendering analysis, we implemented our framework and the concept of Firework Plots. We show the usefulness of the implementation by discussing different use cases and comparing it to related work. Moreover, we conducted multiple benchmarks to show the rendering performance and calculation times.

Contents

CHAPTER 1

Introduction

Interactive visualizations are very important for exploratory data analysis. Especially for large datasets, such tools are needed to develop insights. Therefore, many interactive exploration tools for data experts from different domains already exist. However, often datasets are also of interest to a broad audience. In this case, the standard procedure is that domain experts use their knowledge about the data and their findings to select a meaningful representation for the information and insights they want to present. But this also means that the audience only sees a small part of the actual information, which the domain expert found interesting.

The Austrian Media Transparency Database [Ope] is a good example. It arises from the Austrian *Medienkooperations- und -förderungs-Transparenzgesetz* [Med], which is a law that compels legal entities to announce their advertisement spendings to media organizations. This dataset is of interest to a broad audience because it contains, not exclusively, information of the distribution of Austrian tax money. Moreover, journalists try to find potential political influence on media organizations in this data. However, due to the large size of this database, journalists have to process it for the broad masses by picking interesting aspects. The presentation of the data is mostly supported by simple visualizations, such as bar and line charts. In this case, these visualizations are used to *present* some of the journalists' findings. But they provide no means for a broad audience to perform *exploratory analysis* to discover potentially unexpected information.

This motivates us to make such datasets accessible to a broad audience. However, this leads to two main challenges: to visualize a large dataset with common, and intuitive, techniques and to make this visualization and the large dataset technically accessible to a large audience. In this thesis, we focus on datasets with characteristics similar to the above mentioned Austrian Media Transparency Database as it represents our motivation very well. It can be formally described as a large bipartite graph with weighted relations. One set consists of legal entities and the other set of media organizations. A relation

represents that a legal entity paid money to a media organization and can be weighted by the amount of money.

Summarized, this leads to the following requirements:

1. the visualization should be accessible using commodity hardware and without any special software installation,
2. the visualization and interaction system should be as self-explanatory and understandable as possible for inexperienced users,
3. the visualization should be capable to display thousands of nodes and edges,
4. it should provide an overview of the entire data set
5. it should support common graph analysis tasks as described by Lee et al. [LPP+06] with a focus on topology-based tasks, like detecting clusters.

To fulfil Requirement 1, we decided to implement a web-based approach. Web applications are highly compatible, independent of the underlying hardware, and nothing needs to be installed to run them. This requirement also means to consider the technical limitations of web-based rendering techniques on average consumer-hardware, which we investigate in Chapter 3.

For Requirement 2, we can conclude from the user study of BiCFlows by Steinböck [Ste18] that a more intuitive visualization, than linked lists, should be aimed for the exploratory analysis of large bipartite graphs. Therefore, we decided to use node-link diagrams, which were shown to be easy to understand in multiple user studies (see Chapter 4). However, their main drawback is that they do not scale well with the number of nodes and edges. To solve this problem, we propose a new node-link diagram visualization technique - *Firework Plots* - based on the hierarchical aggregation of nodes, which allows us to achieve Requirement 3. Moreover, with the hierarchical aggregation, we can also generate views of the dataset that can be used as an overview (Requirement 4).

To fulfil the last requirement, Requirement 5, we present exploration tools to drill-down the hierarchy, provide details-on-demand and filtering. With these tools and our visualization concept, topology-based tasks like finding adjacent or accessible nodes as well as finding clusters and connected components can be achieved on multiple levels of abstraction. Moreover, the attribute-based filtering enables users to fulfil attribute-based tasks, and with the hierarchical aggregation, overview tasks can be achieved.

The goal of this thesis is to present a web-based interactive exploration interface for, large, weighted, bipartite graphs. First, we demonstrate the state-of-the-art of web-based rendering of large graphs (see Chapter 3). We compare the most common rendering techniques and libraries that provide APIs for these techniques and conduct multiple benchmarks. This comparison helps us to determine the most promising implementation

in terms of rendering performance. Based on these investigations and the accessibility requirements, we present a novel web-based visualization framework of large, weighted, bipartite graphs that ensures perceptual and interactive scalability. Due to the web-based approach, we have a compatible framework, which runs on consumer hardware. The client-server structure allows moving expensive preprocessing and calculation steps to the server (see Chapter 6). Furthermore, the main part of our visualization framework consists of our visualization concept, called Firework Plots. It uses a node-link diagram as a basic concept and is a combination of hierarchical aggregation, layered force-layouts, semantic zoom and measures to reduce clutter (see Chapter 5).

The contributions of this thesis can be summarized as follows:

- A comparison of web-based rendering techniques and libraries in terms of graph rendering (Chapter 3).
- The concept of Firework Plots to visualize large graphs on multiple levels of detail (Chapter 5).
- The implementation of an interactive web-based framework to visualize and explore large bipartite graphs with more than thousands of nodes and tens of thousands of edges (Chapter 6).

We demonstrate the expressiveness of Firework Plots by showcasing two datasets with different sizes and densities. Also, we quantify the performance gains by our implementation by conducting performance benchmarks and measuring the improvements by the step-wise introduction of measures of the Firework Plots.

CHAPTER 2

Definitions, Terms, and Data Characteristics

This chapter provides definitions and explanations of terms and concepts needed for this thesis. We start with definitions and characteristics of graphs and then specialize in bipartite graphs. At last, we describe the characteristics of the data, which is used in this thesis and further describe the datasets we use for evaluation.

2.1 Graphs

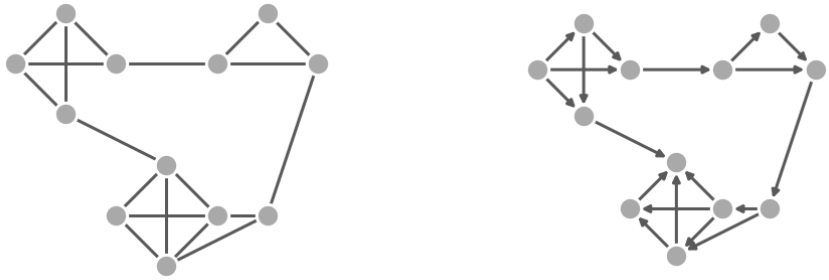
Graphs are a common concept to describe relations between entities. They are widely used in different domains, for example, biology, social network analysis, or finance. We, therefore, want to provide a formal definition:

Definition. A *graph* $G = (V, E)$ consists of a set of *vertices* (or *nodes*) V and a set of *edges* E, such that $E \subseteq V \times V$. The vertices of an edge are called *endpoints*. An edge *connects* its two endpoints.

Graphs can be distinguished between *directed* and *undirected graphs* (see Figure 2.1):

Definition. A *directed graph* is a graph $G = (V, E)$ where the edges $e \in E$ are ordered pairs of vertices such that $e = (u, v)$ for $u, v \in V$ and $(u, v) \neq (v, u)$. The edges in a directed graph are called *directed edges*.
An *undirected graph* is a graph where the edges are unordered pairs of vertices such that $e = \{u, v\}$ and $e = \{u, v\} = \{v, u\} = uv$.

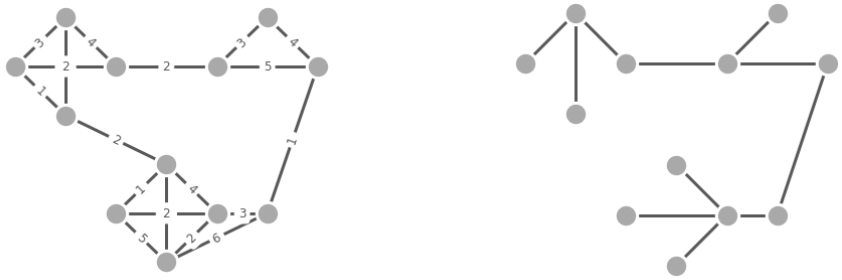
There are different attributes to further describe different types of graphs. A *weighted graph* is a graph where each edge has a weight w for $w \in \mathbb{R}$ assigned. A weighted graph



(a) (b)

Figure 2.1: Example of an undirected graph (a) and a directed graph (b).

is also called a *network*. A *walk* is a sequence of edges where successive edges have a vertex in common. A *cycle* is a walk, which starts and ends in the same vertex and where no edges and nodes are repeated. A graph is *acyclic* if the graph contains no cycles. Figure 2.2 shows an example of a weighted and an acyclic graph.



(a) (b)

Figure 2.2: Example of a weighted graph (a) and an acyclic graph (b).

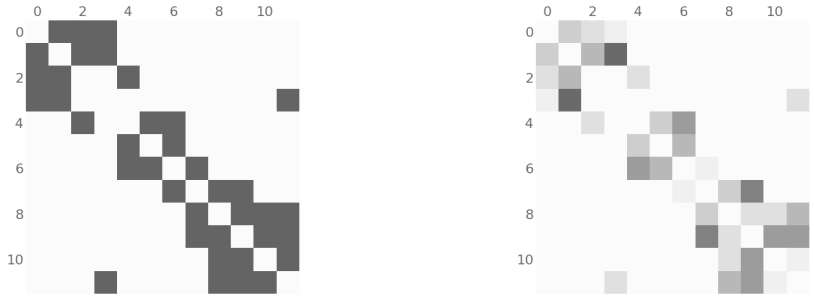
An edge that connects a vertex to itself, i.e. (v, v) or $\{v, v\}$ is called *loop*. If there is more than one edge connecting the same vertices the graph is said to have *multiple edges*. If a graph contains no loops and has no multiple edges it is called a *simple graph*.

An edge and a vertex are *incident* if the vertex is an endpoint of the edge. This means for $e = \{u, v\} \in E$, e and u are incident as well as e and v. Furthermore, the vertices u and v, connected by an edge, are called *adjacent* and u is a *neighbour* of v and vice versa.

A common concept to represent a graph is the *adjacency matrix*:

Definition. The *adjacency matrix* $A = (a_{i,j})$ of a graph $G = (V, E)$ is a matrix where $V = \{v_1, ..., v_n\}$ and $i, j = 1, 2, ..., n$ and the entries are defined as

$$a_{i,j} = \begin{cases} 1, & v_i \text{ is adjacent to } v_j \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$



(a) (b)

Figure 2.3: Adjacency matrix of a graph without edge weights (a) and with edge weights (b).

For an undirected graph, the adjacency matrix is symmetric. If a graph contains no loops, the entries on the diagonal are $a_{i,i} = 0$. The entries in the adjacency matrix of a weighted graph are the corresponding edge weights $a_{i,j} = w_{i,j}$ if there is an edge $e = \{v_i, v_j\} \in E$ with weight $w_{i,j}$. In Figure 2.3, one can see an adjacency matrix of an unweighted graph compared to that of a weighted graph.

To further describe graphs, additional properties exist. Common properties of a graph are the *cardinality* of V and E, i.e., the number of vertices $|V|$ and edges $|E|$. The *density* D of a graph is the ratio of number of edges to the number of possible edges, i.e. for a undirected graph

$$D = \frac{2|E|}{|V|(|V|-1)} \tag{2.2}$$

and for an directed graph

$$D = \frac{|E|}{|V|(|V|-1)}. \tag{2.3}$$

The *degree* of a vertex denotes the number of edges connected to the vertex.

For this thesis, we also need to describe the concept and terms of *graph clustering*. The goal of *graph clustering* is to partition a graph based on the edges into groups, also called *clusters*. Moreover, graph clustering is often applied to weighted graphs, and, therefore, can also be based on edge weights. Clusters are subgraphs, and subgraphs are formally defined as follows

Definition. A *subgraph* $H = (V', E')$ of $G = (V, E)$ is a graph where $V' \subseteq V$ and $E' \subseteq E$.

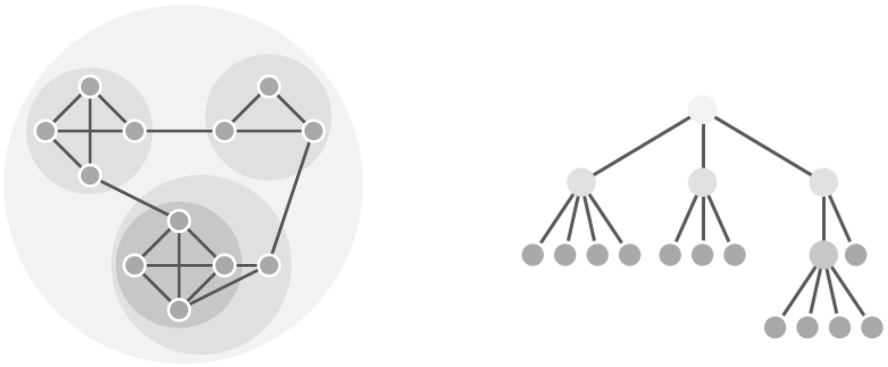
A subgraph is a *connected component* if any two vertices of the subgraph are connected to each other by a walk and if no vertex of the subgraph is connected to any other vertex of the original graph.

Besides, we also need the following definitions to further describe graph clustering:

Definition. A graph which is undirected and acyclic is called a *forest*. Furthermore, if the forest is connected, it is a *tree*. If there is a single node which can be fixed as *root*, the tree is a *rooted tree*.

A *leaf* is a node in a tree which has degree one. The *level* of a vertex in a tree is the minimum number of edges to get from the root to the node. If we refer to a certain level of a tree, we mean all vertices with this level. If a vertex u is incident to a vertex v and v has a lower level than u, then v is called *parent* of u and u is a *child* of v. A *descendent* of v is every vertex, which is a child of v, or, recursively, a child of its children.

If a clustering algorithm is applied recursively, a *graph hierarchy*, also called *compound graph*, is obtained. If clustering is used to generate a graph hierarchy, it is often referred to as *hierarchical clustering*. The graph hierarchy is a rooted tree where the root represents a cluster containing all nodes of a graph. Each level represents a more detailed clustering. The leaves of a graph hierarchy represent the original graph nodes. In Figure 2.4, the results of the hierarchical aggregation of a graph can be seen in a graph view (Figure 2.4a) as well as the resulting hierarchy tree (Figure 2.4b). The smaller nodes with a white border represent the original nodes or leaves. In the context of visualization, we often refer to cluster nodes as *meta nodes*.



(a) (b)

Figure 2.4: Hierarchically clustered graph where clusters are encapsulated by circles (a) and the corresponding graph hierarchy (b).

We also want to provide the definition of a *path-preserving hierarchy* by Archambault et al. [AMA09]:

Definition. A *path-preserving hierarchy* is a specific type of graph hierarchy that must respect two properties:

1. *Edge conservation:* An edge exists between two meta nodes m_1 and m_2 if and only if there exists an edge between two leaves in the input graph l_1 and l_2 such that l_1 is a descendant of m_1 and l_2 is a descendant of m_2.

2. *Connectivity conservation:* Any subgraph contained inside a meta node must be connected.

The definition of graph size or more specifically, whether a graph is small or large, is not easy, if not impossible. Most publications use terms like 'small', 'large' and 'complex' to describe graphs, however, there is no common definition what these terms mean. This is the reason why Yoghourdjian et al. [YAD+19] did a survey of 152 empirical studies on graph visualization and also conducted a user study to gain a better understanding of these terms in the context of human perception. In the end, they found the categories in terms of graph size as in Table 2.1 and in terms of graph density based on the linear density

$$D = \frac{|E|}{|V|} \tag{2.4}$$

in Table 2.2.

Category	**# of Nodes**
Small	≤ 20
Medium	$[21, 50]$
Large	$[51, 200]$
Very Large	> 200

Table 2.1: Categories of graph size according to Yoghourdjian et al. [YAD+19].

Category	**Linear Density**
Tree-like & Disconnected	$[0, 1.0]$
Sparse	$[1.0, 2.0]$
Dense	$[2.01, 4.0]$
Very Dense	> 4.0

Table 2.2: Categories of graph density based on linear density according to Yoghourdjian et al. [YAD+19].

2.1.1 Bipartite Graphs

Bipartite graphs are special types of graphs where the nodes of the graph can be partitioned into exactly two distinct sets of nodes such that there are only edges with one endpoint in each set. A formal definition is given as follows:

Definition. A simple undirected graph $G = (V, E)$ is called *bipartite* if and only if $V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset$ and $uv \in E \Rightarrow u \in V_1, v \in V_2$.

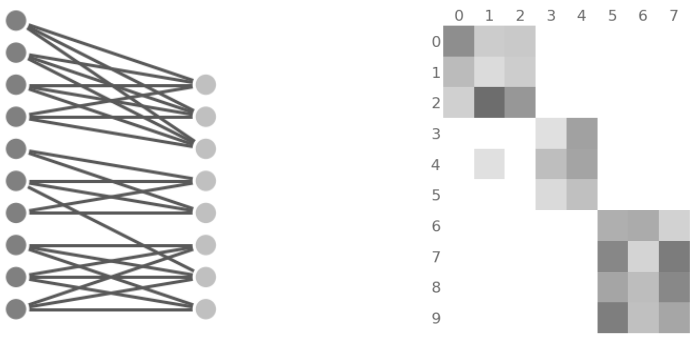
Moreover, a bipartite graph is an acyclic graph, and an acyclic graph is a bipartite graph. Since there are no connections between nodes within a set, the *biadjacency matrix* can be defined to represent a bipartite graph as a matrix.

Definition. Given a bipartite graph $G = (V, E)$ such that $V = V_1 \cup V_2$. The *biadjacency matrix* $B = (b_{i,j})$ of G is a matrix where $V_1 = \{u_1, ..., u_n\}$, $V_2 = \{v_1, ..., v_m\}$ and $i = 1, 2, ..., n$, $j = 1, 2, ..., m$ and the entries are defined as

$$b_{i,j} = \begin{cases} 1, & u_i \text{ is adjacent to } v_j \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

Bipartite graphs can be weighted graphs as well. The entries in the biadjacency matrix of a weighted bipartite graph are then the corresponding edge weights $b_{i,j} = w_{i,j}$ if there is an edge $e = \{u_i, v_j\} \in E$ with weight $w_{i,j}$. Figure 2.5 shows an example of a bipartite graph as well as the weighted biadjacency matrix.

For bipartite graphs, special clustering algorithms, called *biclustering*, exist, which cluster the rows and columns of the biadjacency matrix at the same time. The resulting clusters are also called *biclusters*. As for general graphs, biclustering algorithms can be applied recursively to obtain a graph hierarchy.



(a) (b)

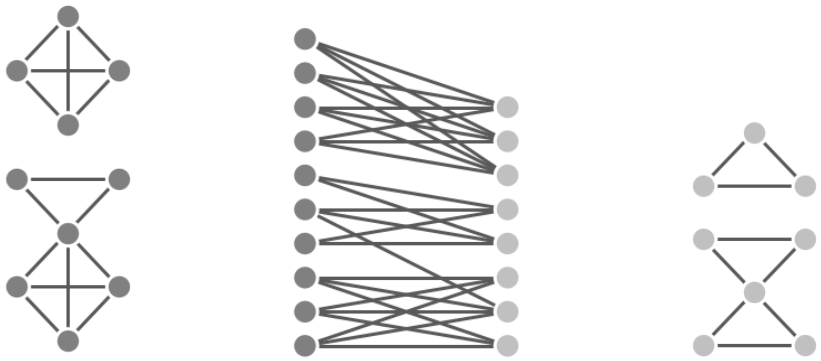
Figure 2.5: Bipartite graph with parallel layout (a) and the corresponding weighted biadjacency matrix (b). Biclustering and matrix reordering was applied before to detect clusters and reduce edge crossings.

Bipartite projections of bipartite graphs can be used to compress information. The idea is to project one set of vertices on the other one based on their relations. Such a projection onto one of the sets is also called *one-mode projection*. Consequently, the underlying bipartite graph is often referred to as *two-mode projection*. A formal definition can be given as follows:

Definition. A *bipartite projection* of G on V_2 is a unipartite graph $G' = (V_2, E')$ where $(v_i, v_j) \in E'$ if there is any $u_k \in V_1$ such that $(v_i, u_k) \in E$ and $(v_j, u_k) \in E$.

A bipartite projection can also be a weighted graph where, for example, the number of vertices of V_1 through which vertices of V_2 are connected is used as edge weight.

Figure 2.6b illustrates the original graph (two-mode projection) and Figure 2.6a and Figure 2.6c show the respective bipartite projections on each set.



(a) (b) (c)

Figure 2.6: Bipartite graph with parallel layout (b) and the corresponding bipartite projections on each set (a, c).

2.2 Data Characteristics

In the real world, we have datasets that often describe relations between entities. For such datasets, graphs are a suitable concept to formalize them. However, datasets often hold more information that cannot be represented by the definitions and terms of the previous paragraphs. We, therefore, need to explain additional terms, which are used in data analysis but not in graph theory. In the following paragraphs, we only focus on datasets, which can be formalized as graphs.

The terms *one-mode* and *two-mode* are also used to describe characteristics of a dataset similarly as for bipartite projections. A two-mode dataset consists of two distinct sets such that there are only connections from one set to the other and not within the entities of one set. In contrast, datasets which only have one type of nodes can be called *one-mode* data. Sometimes, a two-mode dataset is projected on one of the sets, resulting in a one-mode dataset, to compress the information of the dataset. For example, a product-consumer dataset, which we assume to be bipartite, could be projected on the set of consumers. The resulting dataset would be a one-mode dataset only containing the consumers and connections between consumers if they, according to the original dataset, bought the same product.

Datasets which develop over time are *dynamic*. The corresponding type of visualization is called *dynamic graph visualization*. The counterpart of dynamic datasets are the more common *static* datasets.

When working with a dataset, clustering can be applied not only like graph clustering based on the relations between the entities but also based on additional attributes in the

dataset. In the following of this thesis, we refer to clustering based on the underlying graph structure as *topology-based* clustering and clustering based on additional attributes of the dataset as *attribute-based* clustering.

In this thesis, we focus on very large static weighted two-mode datasets. In the next subsections, we will introduce the datasets that were used for the evaluation and meet these requirements.

Media Transparency Database

The database arises from the *Medienkooperations- und -förderungs-Transparenzgesetz* [Med], which is a law that was passed by the Austrian government in 2011. Legal entities (Rechtsträger), which are under the control of the Austrian Court of Audit, have to announce their advertisement spendings (Euro) to media organizations (Medium) per quarter (Quartal). The law has multiple paragraphs (Bekanntgabe), which prescribe the publication of different types of spendings. At the end of each quarter, the data of the previous quarter is published online [Ope]. In Table 2.3 the column labels and three example entries (rows) can be seen. The column Leermeldung is a boolean that is true (1) only if an organization has no advertisement spendings to announce in this quarter. Entries which have 1 in Leermeldung are, therefore, discarded.

Rechtsträger	**Quartal**	**Bekanntgabe**	**Leermeldung**	**Medium**	**Euro**
Österreichischer Rundfunk	20184	2	0	www.youtube.at	5831.00
ÖBB-Werbung GmbH	20184	2	0	Österreich - oe24	12881.72
Ärztekammer für Wien	20184	2	0	gesund & fit	31000.00

Table 2.3: Example entries of the Media Transparency Database [Ope].

The database contains, beginning from the third quarter of 2012 to the fourth quarter of 2019, more than 70000 entries with advertisement spendings. Although this dataset is dynamic, we do not focus on this property. Formalized as a graph, this means it has more than 70000 edges with 5925 nodes from which 1393 are legal entities and the other 4532 are media organizations. However, in our case, we do not consider the temporal attribute of the dataset. This means that links between two entities, although they represent relations at different times, are merged into one link. This results in 17780 data links. Still, the temporal attribute is part of the dataset and can, therefore, be used for other purposes in our framework.

This dataset is of potential interest to a large audience and therefore also inexperienced users because it provides insights into the distribution of Austrian tax money and connections between important political entities and selected media cooperations. In terms of the classification by Yoghourdjian et al. [YAD+19] this graph can be categorized as a very large and dense graph having a linear density of $D \approx 3.0$.

Medical Dataset

Another dataset has a medical context and was provided to us by data scientists. Since we do not have the permission to describe any details or show any insights in this thesis, we will only focus on its properties.

This dataset provides columns which specify the source and the targets of the relations that also define the two sets of a bipartite graph. Moreover, there is one column defining the respective edge weights as well as three additional columns which are treated as link attributes. It consists of more than 88367 links which define the relations between 4416 nodes. The two sets are unbalanced where 1147 nodes belong to one set and the remaining 3269 nodes to the other one. Therefore, we can categorize it, according to Yoghourdjian et al. [YAD+19], as a very large and very dense graph with a linear density of $D \approx 20.01$.

CHAPTER 3

Graph Drawing on the Web

This chapter aims to give an overview of the possibilities and limitations of rendering large graphs on the web. The state-of-the-art in web rendering is to use JavaScript on the client-side. Due to the rising popularity of web applications, different types of drawing techniques were developed and visualizations on the web got more efficient which lead to the development of many different libraries.

A benchmark suite was set up, which is used to compare the performance of the three standard drawing techniques (SVG, Canvas, and WebGL). Moreover, available libraries of the best performing rendering technique were investigated and compared.

3.1 Web-based Rendering Technologies

SVG [SVG], Canvas [Can], and WebGL [Web] are the three state-of-the-art web-based rendering technologies. They are completely different types of techniques and, therefore, do not only differ in their performance but also their intended application and usability.

SVG stands for Scalable Vector Graphics and is based on the Extensible Markup Language (XML). It is easy to use and generates high-quality 2D vector graphics. It also has good support for interactions and animations. However, when using SVG, many static DOM elements are created, which have a significant influence on the rendering performance, especially for large, dynamic, and interactive visualizations. Since an SVG scene consists of many DOM elements, the intended use was for small scenes with a fixed number of elements. In contrast to animating elements, adding and removing elements from the scene, subsequently, is very expensive.

Another well-known technique is Canvas. Its purpose is to draw 2D graphics using JavaScript. It is bitmap-based and is only suited for simple renderings using 2D objects like lines, rectangles and circles. Nevertheless, depending on the size of the HTML Canvas it can perform better than SVG for such simple drawings, as it does not generate any

DOM elements. This comes with the drawback of a worse quality compared to SVG. Rendering a dynamic scene with this technique only works for a small number of objects since every change in the scene requires a complete repaint.

WebGL has the significant advantage that it supports rendering on the GPU what further improves the performance. Additionally, it offers 3D rendering and more freedom in scene design and rendering of complex meshes. It is intended to run a constant rendering loop to efficiently render dynamic scenes, but a redraw of the scene can also be initiated on-demand for static scenes. One important issue to be mentioned is that rendering lines (GL.Lines) with different linewidths are not supported. WebGL is based on OpenGL ES 2.0/ES 3.0 and its specification [LL19] states that the maximum and the minimum line width is only allowed to be 1.0. However, there are many use cases where lines with linewidth are important, as it is the case in this thesis for rendering weighted graphs. For the benchmarks, we used Three.js [Dan12] as a stand-in for WebGL.

3.2 Benchmark Setup

For our benchmarks, we use `window.requestAnimationFrame(callback)`, which calls the callback function passed to it before the next repaint. We pass the render function of the corresponding technique or library to be called. The `requestAnimationFrame` function is synced with the displays refresh rate. This means that a dynamic application typically runs at 60 FPS, which means $\sim$ 16.7ms per frame. However, if rendering gets too expensive, repaints are not delayed but entirely skipped. This means that the repaints might constantly jump between 16.7 ms and $2 \times 16.7 = 33.4$ ms. For the benchmarks, we collect 100 frames, remove the first ten frames and average across the remaining 90 frames.

Depending on the type of benchmark we generated different scenes. In one test scene, we only render 2D circles at random positions where the number of circles is further increased to determine the number of possible elements within a scene. Another type of test scene consists of random graphs generated using the `gnm_random_graph()` function of the Networkx [HSS08] library, where the number of nodes is fixed at 4500 to generate up to 10M links between those nodes. The positions of the nodes are again chosen randomly and links between them are drawn as defined by the generated graph. The described test scenes are also used in dynamic variations where the positions of the scene elements are changed at every render call to compare the update performances and therefore their ability to render dynamic scenes. In Figure 3.1 a test scene with 4500 nodes and 500 links is shown. Additionally, the initialization times were measured. To do this, the initialization step was executed 30 times for each test instance, then the first five runs were removed and the average of the remaining 25 measurements were calculated. All approaches, except SVG, are rendered on a canvas with a size of 700×900 pixels.

The performance benchmarks were executed on a Windows 10 machine with a 4.3 GHz AMD Ryzen 5 processor with 16 GB DDR4 RAM and an Nvidia GTX 1080 graphics card with 16 GB memory. As a browser, we used Chrome with the 64-bit version 79.0.3945.130.

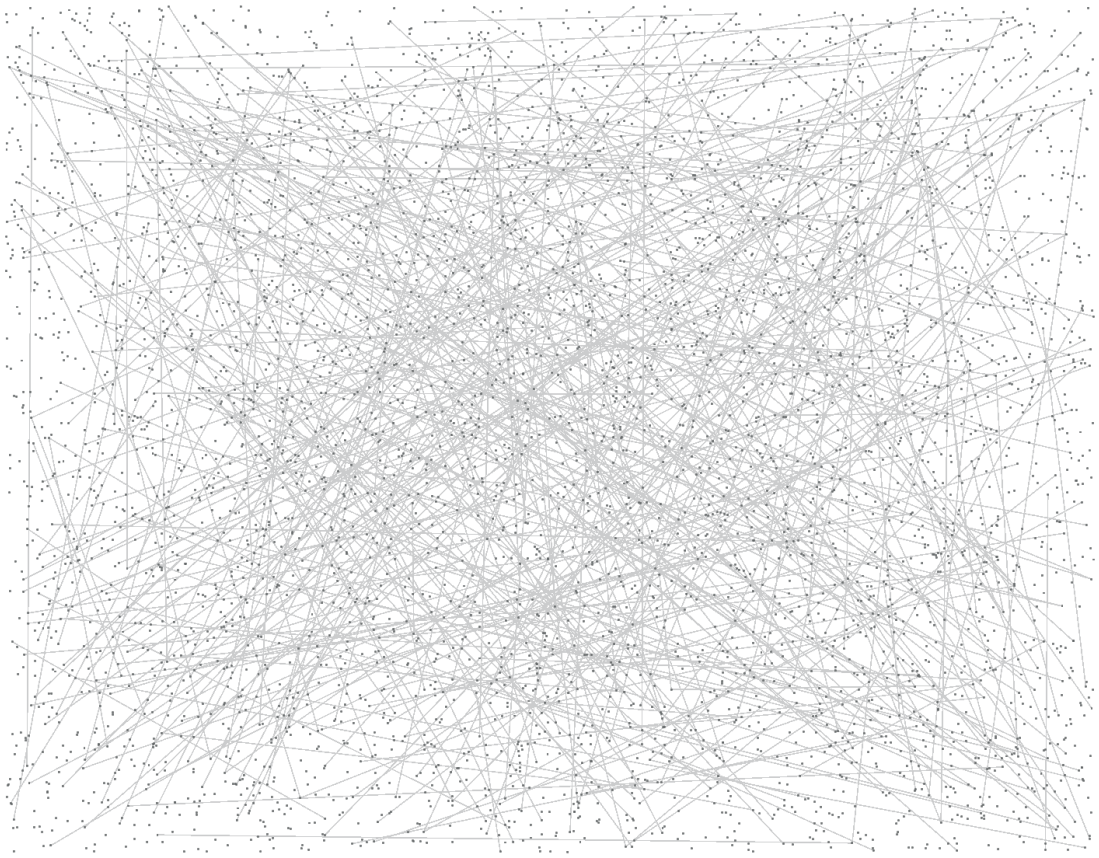
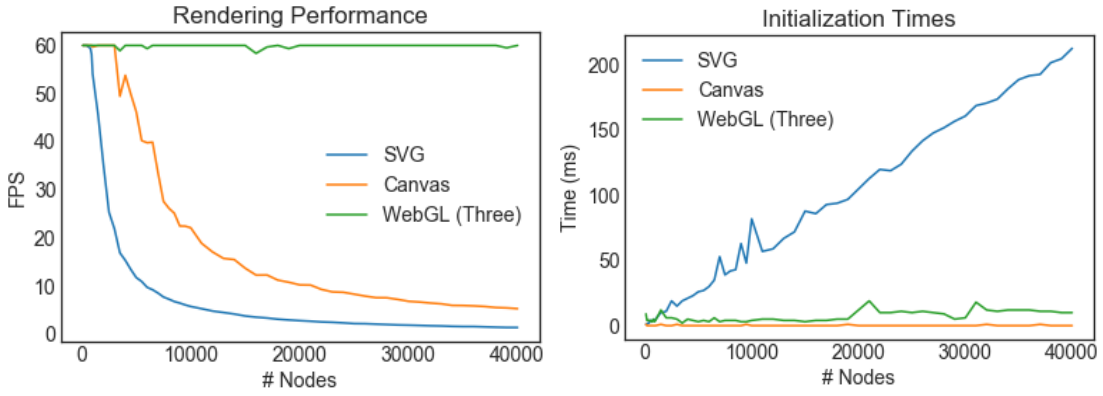


Figure 3.1: Test scene with 4500 nodes and 500 links.

3.3 Benchmarks of Rendering Technologies

To compare the performances of SVG, Canvas, and WebGL a dynamic test scene was used where only circles with dynamic positions are rendered. The positions of the nodes are updated at each render call. For the SVG scene `SVG Circle` was used, for canvas the HTML canvas `arc()` function was used and in the WebGL scene `GL.Points` were rendered with a circle texture. We chose this setup to determine which number of elements is possible to render with animations and to trigger repaints of the browser. Figure 3.2 shows two plots of the results of our benchmarks. Figure 3.2a shows the rendering performance in terms of FPS and Figure 3.2b depicts the initialization times in milliseconds to set up the scene and its scene objects.

It is obvious that in terms of rendering performance, Canvas performs better than SVG and WebGL performs by far better than SVG, as well as Canvas, even for thousands of nodes in the scene. In terms of initialization times, the plots confirm that creating DOM elements for SVG is very expensive and, therefore, initialization time increases with the number of nodes. For initialization of a Canvas scene, only the respective Canvas



(a) (b)

Figure 3.2: Comparison of rendering performance of SVG, Canvas, and WebGL in terms of FPS (a) and initialization times in milliseconds (b) with an increasing number of circle nodes, but no edges.

DOM element needs to be created. As the initialization is independent of the number of elements, the initialization time of Canvas is constantly $< 1ms$.

These benchmarks show that WebGL is the most efficient rendering technique on the web if the focus is on rendering many elements and not high-quality results. However, with an increasing number of elements, the initialization time needs to be considered. Still, WebGL defines the current limitations of rendering a large amount of data on the web and we will further inspect it in the next section.

3.4 WebGL-based Graph Rendering Libraries

Since WebGL performs far better than the other techniques, in the past years, many JavaScript libraries were developed that support WebGL. These libraries define different types of APIs to the WebGL calls. Their main difference, therefore, is their target audience and the encapsulating code to make these calls. The most common visualization libraries using WebGL are P5 [McC], Sigma [JP], Stardust [RLH17] and Three.js [Dan12]. Of these only Sigma and Stardust are specialized in graph rendering.

P5 [McC] is a library developed for designers and beginners. Its API is simple and easy to use. The default renderer used by P5 is Canvas but WebGL can be explicitly specified, which was done for the benchmarks. Due to the target audience of P5, their focus lies on small, animated, scenes and their API is restricted to drawing the objects as defined by them.

Sigma [JP] is a library explicitly for graph rendering. The default renderer is, as well as for P5, Canvas but can be opted to be WebGL as done for the benchmarks. Its API is very simple and easy to use and graph visualizations are generated very fast. However,

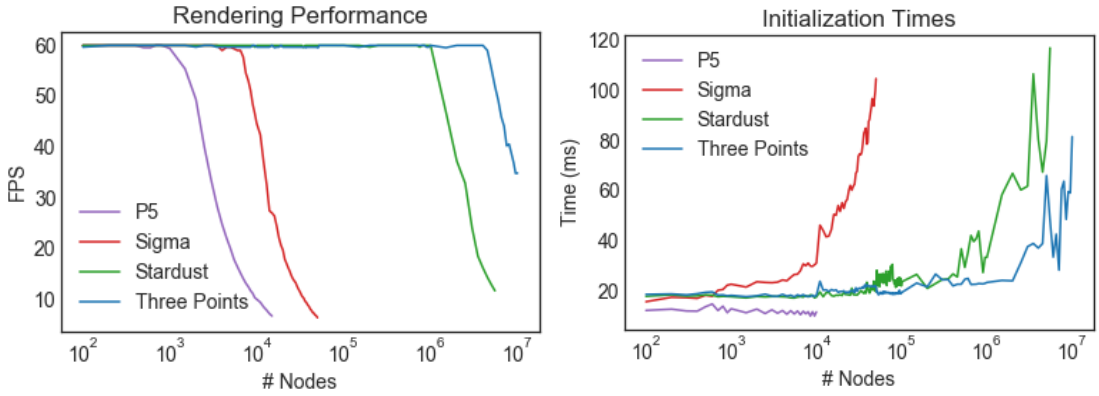
this simplicity is gained by making many restrictions, as rendering only supports circles with different colours and sizes, and lines with uniform line width.

Stardust [RLH17] is entirely focused on rendering with WebGL. They encapsulate the WebGL calls such that the user does not need to manipulate any shaders or buffers but can still define *marks* that can be rendered. Also, the developer of Stardust implemented a workaround to display lines with variable linewidth as well. The manipulation of shaders and buffers - and therefore custom visualizations outside the box - are not possible.

Three.js [Dan12] is implemented such that it allows an easy start with WebGL but also offers to go into details of WebGL without any restrictions. Moreover, the developers of Three.js implemented a shader [Lin] that allows rendering lines with adjustable linewidth as well. Even though the API offers an easy start with WebGL, it is still far more complex than the other libraries. This is because one has to create and fill buffers, as well as specify materials. As other libraries do this automatically they have to build more code around this buffer manipulation which is done in the background and might come with drawbacks in performance.

Figure 3.3 shows the results of our first test. The purpose of this test was to determine how many elements are possible to render. We used a static scene in which only nodes are rendered. Although it is a static scene, we force the redraws for each library to compare performances. Note that the number of nodes is plotted on a logarithmic scale. All scenes were rendered using WebGL. If a line ends, this means that measuring the performance of this library was stopped. This was done because it can be assumed that the performance is not going to get better and further measurements are not going to show more insights, but would unnecessarily cost computation times. In the plot we refer to `Three Points`, as this is the standard Three.js implementation to render points [Poi] and we are going to discuss different shader implementations, including this one, throughout this thesis. P5 performs worst in terms of rendering performance but has the lowest and most reliable initialization times. This is due to the architecture of P5. As the default renderer is Canvas, the setup does not change when switching to the WebGL renderer. This means the initialization steps only require the creation of a Canvas DOM element and for each redraw, all elements of the scene are newly defined and drawn to the Canvas. Sigma performs a bit better in terms of rendering, but initialization times rise very fast. The performance of Stardust and Three.js are about the same level but Three.js still outperforms Stardust, both in terms of rendering and initialization.

As dynamic scenes are required for this thesis, we conducted the same test again, but with updates in the positions of each node in each render call. This only influences the rendering performance and not the initialization times. Therefore, only the FPS are shown in Figure 3.4. The node position updates were done by updating the positions of the nodes in the buffer. We can see that all libraries perform very similar to the previous benchmark except Stardust. Stardust is not optimized in terms of data updates, which shows that the direct buffer manipulation in Three.js is more efficient. At 10^7 nodes it can be seen that the iteration and updates also cost approximately 10 FPS with this number of elements compared to the static scene.



(a) (b)

Figure 3.3: Comparison of rendering performance of different WebGL libraries in terms of FPS (a) and initialization times in milliseconds (b) with an increasing number of nodes with static positions.

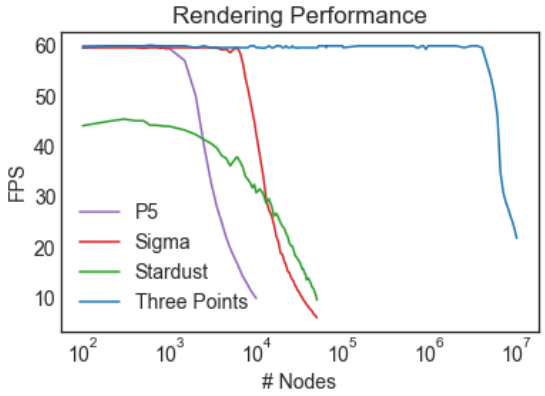
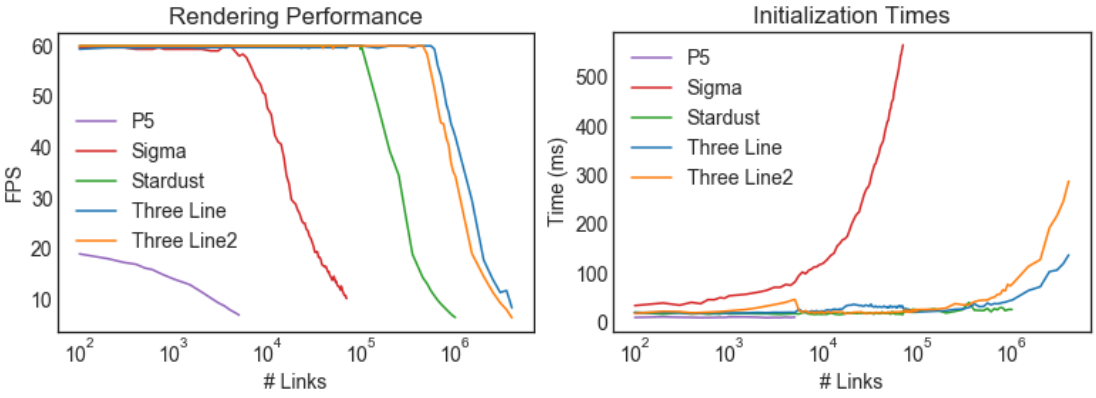


Figure 3.4: Comparison of rendering performance of different WebGL libraries in terms of FPS with an increasing number of nodes with dynamic positions.

In the same manner, we did these static and dynamic tests with scenes including lines between the nodes to simulate our requirements for graph rendering. The results of these benchmarks can be seen in Figure 3.5 and 3.6. To determine how much influence the rendering of lines with linewidth has on the performance, we also compared normal line rendering (`Three Line`) and the workaround which renders lines with linewidth (`Three Line2`). We can see that, overall, the performance of the different libraries looks very similar to the previous ones in both cases, i.e., static and dynamic. The performance of P5, however, suffers a lot from the fact that our starting point is at the sizeable number of 4500 nodes and the links are just added on top of it. Moreover, a small loss in performance using lines with linewidth is visible.



(a) (b)

Figure 3.5: Comparison of rendering performance of different WebGL libraries in terms of FPS (a) and initialization times in milliseconds (b) with 4500 nodes and an increasing number of links with static positions.

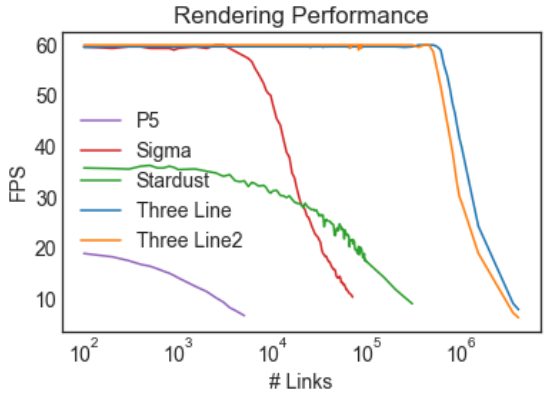
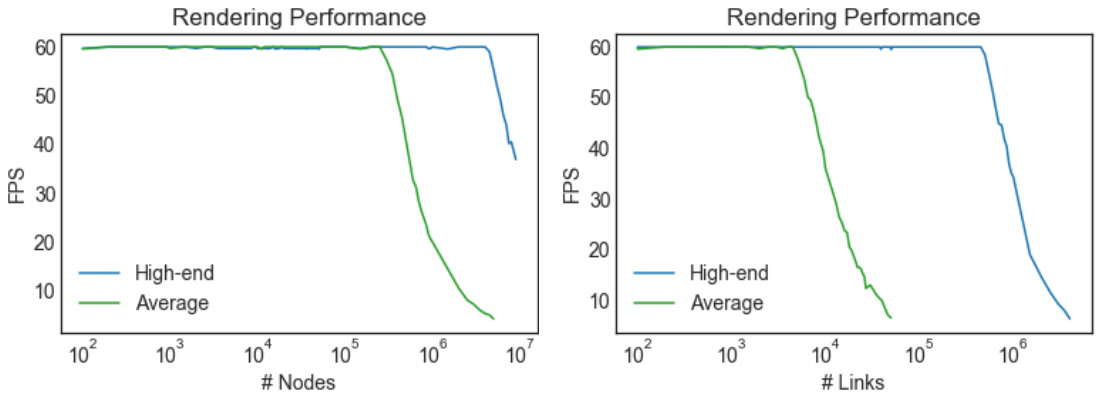


Figure 3.6: Comparison of rendering performance of different WebGL libraries in terms of FPS with 4500 nodes an increasing number of links with dynamic positions.

The presented results show rendering performance on a high-end consumer computer and do not represent the performance on an average consumer setup. Therefore, the static benchmarks for nodes and lines were also conducted on a machine that better represents such machines. This helps to get an overview of the influence different hardware can make. The hardware representing an average setup is a 64-bit Windows 10 machine with an Intel Core i5-6300U processor with 8 GB RAM and the built-in HD Graphics 520 graphics card with 4 GB memory. Figure 3.7 shows the rendering performance of the different hardware configurations using the static benchmark setup. The results with an increasing number of nodes are shown in Figure 3.7a and those with an increasing number of links in Figure 3.7b. We refer to the better hardware, which was used for all

other benchmarks in this chapter, as `High-end`, and to the hardware better representing the consumer hardware as `Average`. The plots show very well that there is a significant difference, especially for rendering lines, due to the underlying hardware.



(a) (b)

Figure 3.7: Comparison of rendering performance of a high-end and an average consumer machine in terms of FPS. Rendering performance with an increasing number of nodes (a) and performance with 4500 nodes and an increasing number of links (b) are shown.

The benchmarks in this chapter do not just show differences of available libraries based on WebGL but, more importantly, the current limitations in rendering graphs on the web. Moreover, the last results on a machine better representing the average consumer hardware show that the rendering performance already starts to become a problem with less than 10^4 links between 4500 nodes. To be precise, our measurements show that beginning with 6000 links the FPS are already below 60 and the end of smooth rendering is reached. If we want to render larger graphs on machines like this, or even weaker ones, we need visualization strategies. This is one of the main motivations for our visualization concept, which is described in Chapter 5.

CHAPTER 4

Related Work

Although the focus of this thesis is on the visualization of bipartite graphs, many concepts of general graph visualization apply to special types of graphs like bipartite graphs. Particularly, the basic visualization concepts, interaction techniques, and approaches in handling large data sets are of interest. To gain an insight into existing work and its applicability for the problem statement of this thesis, we first discuss general graph visualization techniques. Afterwards, we go into detail on existing visualizations of bipartite graphs.

4.1 Graph Visualization

Graph visualizations are used in many different domains like biology [ADWM04, GBMK08, FHK+09], social network analysis [LCL+09, HB05], finance [CGK+07], software engineering [BDL05] and many more. Their purpose is to support the user in the exploration and analysis of the underlying data set, its entities and the relations between them. The goal is that the user understands the data globally as well as local structures and connections in the data. The user should be assisted in finding insights like clusters of highly connected entities.

We can distinguish between three different kinds of visual representations of graphs. These are *Node-Link Diagrams*, *Adjacency Matrices*, and combinations of these two, called *Hybrid Visualizations*. Small examples of each category are depicted in Figure 4.1. Related work of these visualizations will be discussed in the next sections, including the respective interaction techniques and approaches of handling the scalability issue for large graph visualizations.

(a) (b) (c)

Figure 4.1: The three basic types of graph visualization from Henry et al. [HFM07]. (a) illustrates a node-link diagram, (b) an adjacency matrix, and (c) a hybrid visualization.

4.1.1 Node-Link Diagram

A node-link diagram consists of nodes and edges, connecting those nodes. Figure 4.1a visualizes a small example of a node-link diagram. Nodes represent the entities of a dataset and the edges represent the relations between these entities. If two entities have a relationship defined in the dataset, the corresponding nodes in the node-link diagram are connected by a link.

The basic concept of node-link diagrams may be adapted depending on the dataset and the type of graph it can be represented as. If the data contains directed relations, it makes sense to add concepts to the diagram such that those relations are visualized. Moreover, the size and colour of the elements are often used to represent data attributes. In this thesis we do not focus on directed graphs, therefore, we do not further discuss this aspect.

Layout

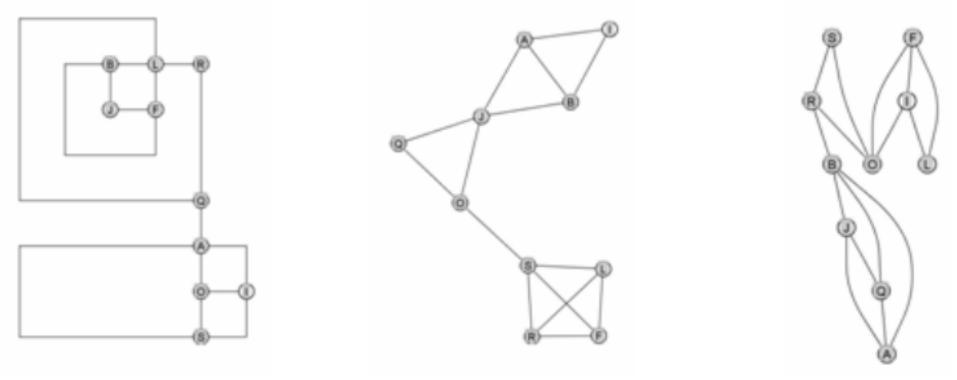
Graph layouts are an important and widely researched field due to their importance in graph readability. Especially for larger graphs, it becomes impossible to manually define graph layouts. Even for small graphs, users tend to give up on layout tasks quickly [DLF+09]. We classify graph layouts in the following three categories:

- **Force-directed Layouts**
  This graph layout technique is based on physical simulations. Forces can be assigned to nodes and edges which may attract or repel each other. The drawback of force-directed layouts is that they need a lot of computation and do not scale well for large graphs. Algorithms of well-known force-directed layouts are, for example, the Fruchterman-Reingold [FR91] and the Kamada-Kawai [KK89] algorithm. An example of a graph with a force-directed layout can be seen in Figure 4.2b.

- **Constrained-based Layouts**
  Constrained-based layouts usually are force-directed layouts where constraints

for node positioning can be defined. Such constraints could be, for example, horizontal or vertical positioning of nodes or overlap avoidance. Figure 4.2a shows a constrained-based layout (more specifically an orthogonal layout) by Pohl et al. [PSD09], where edges only consist of vertical and horizontal lines. Since force-directed layouts do not scale well, adding constraints to them results in even longer execution times.

- **Hierarchical Layouts**
  Hierarchical layouts, also called layered layouts, are usually used for directed graphs as can be seen in Figure 4.2c. The nodes are positioned at different layers depending on the connections.



(a) (b) (c)

Figure 4.2: Graph layouts prepared for their user study by Pohl et al. [PSD09]. (a) shows a constrained-based layout, (b) a force-based layout, and (c) a hierarchical layout.

Ham et al. [vR08], as well as Dwyer et al. [DLF+09], conducted user studies where participants got the task to adapt the layout of a given graph such that they find it appealing. These user-generated layouts were then compared to common layout algorithms showing that the users perform different tasks on force-directed methods as good as on own layouts. Orthogonal and circular layouts, however, turn out to be worse for the given tasks. In the user study by Pohl et al. [PSD09], eye-tracking is used to compare the readability of small, force-directed, orthogonal, and hierarchical layouts. Their results also indicate that force-directed layouts should be preferred, at least for small graphs. To overcome the limitations of force-directed layouts, there are approaches in outsourcing the calculation to the GPU [GHGH09].

A variation of node-link diagrams are arc diagrams. Nodes are located along one axis and the links between the nodes are drawn as arcs. This has the advantage that there is a place for labels for every node. Also, if the nodes are reordered, cluster structures can be seen very well. However, the overall structure of the graph is better visible using a 2D graph layout.

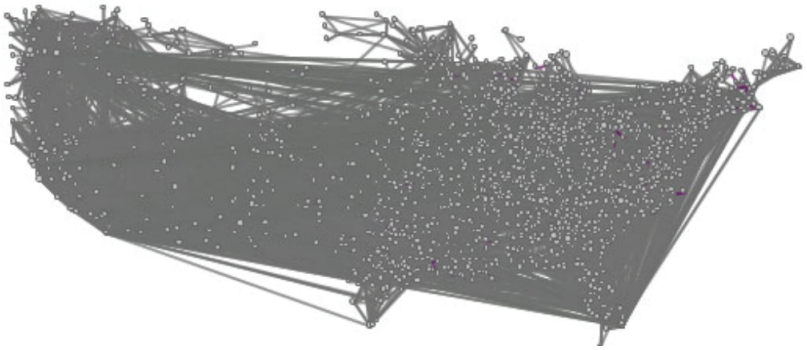


Figure 4.3: Graph showing the US migration without any measures to reduce clutter or improve readability by Cui et al. [CZQ+08].

Measures For Large Graphs

The user studies by Ghoniem et al. [GFC05] and Keller et al. [KEC06] show that with increasing graph size the readability is decreased. Therefore, attempts to overcome this problem and improve readability for large graphs are needed. Figure 4.3 shows an example of a large, cluttered, graph representing the US migration. The most popular approaches to reduce visual clutter are *Graph Aggregation* and *Edge Bundling*.

Graph Aggregation

The concept of graph aggregation is to merge multiple nodes into only one meta-node to form a compound graph. This reduces the number of nodes and edges in the visualization, therefore the amount of clutter, and allows the exploration of clusters and inter-cluster relationships. Such aggregation can either be predefined as in hierarchical graphs or has to be computed. In case a dataset does not come with such an aggregation, it can be generated using clustering algorithms. If applied recursively, an hierarchical aggregation can be created. Clustering algorithms can be distinguished into *attribute-based* and *topology-based* approaches.

With *attribute-based* clustering algorithms, clusters are generated based on vertex attributes defined in the data. Xu and Wunsch [XW05] wrote a survey on attribute-based clustering algorithms. One problem of this type of algorithms is the decision on which features are relevant for the clustering. In GrouseFlocks [AMA08], users can create meta-nodes, which leads to a graph hierarchy, based on data attributes. They use pattern matching, i.e., splitting the selected data into matched and unmatched, based on an specified attribute, and category splitting if an attribute represents a category. These two separation methods, applied at every hierarchy level, lead to many possibilities to form a graph hierarchy. However, this approach requires a lot of user input to generate a deep graph hierarchy, and it might take a lot of time to create a graph that fulfils the expectations of a user. Another problem in the context of node-link diagrams is that two

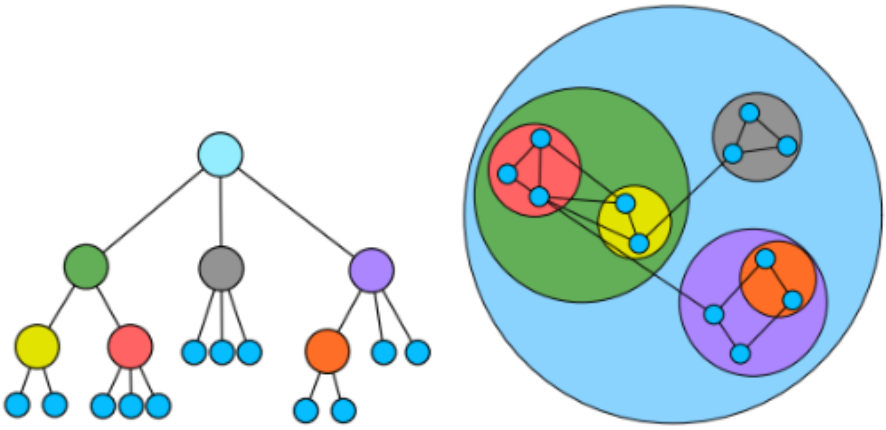
nodes, although they are highly connected through multiple other nodes, might be in different clusters due to some vertex attributes.

Topology-based approaches, on the other hand, try to detect groups of nodes that are highly connected and at the same time reducing the outgoing edges of the group. Forunato [For10] published a survey on the different types of topology-based clustering algorithms. Balzer and Deussen [BD07] use topology-based clustering to avoid altering the graph structure of the layout. Another approach is TopoLayout by Archambault et al. [AMA07]. They use different algorithms to detect different kinds of graph structures and then use this information as constraints for their layout algorithm. We are interested in highly connected clusters of nodes. Therefore, we use a topology-based clustering algorithm. Moreover, we use an implementation of agglomerative clustering based on the publication of Newman et al. [New04]. Agglomerative clustering starts from the bottom where every node is its own cluster. Based on a quality function clusters are merged. This process guarantees that nodes, which are in the same cluster, are connected.

Figure 4.4a shows an example of a graph hierarchy by Archambault et al. [AMA08], where the small dots represent the original nodes, which are the leaf nodes of the hierarchy, and the larger nodes represent the clusters. The visualization in Figure 4.4b shows their approach in graph hierarchy visualization. This approach and its successor, TugGraph [AMA09], draw the graph hierarchy superimposed on top of the node-link diagram using circular enclosures. Batagelj et al. [BBD+11] present another approach, which renders the graph hierarchy superimposed on top of the nodes using rectangular enclosures. In contrast to GrouseFlocks [AMA08] and TugGraph [AMA09], they replace nodes of a cluster with a meta-node and only show the subnodes and its enclosure if the cluster is expanded by the user. A drawback of the approaches using superimposed enclosures is that the graph structure is altered by making links longer or shorter such that they fit into the enclosures which might distort the actual relations within the graph. To overcome this drawback, Balzer et al. [BD07] use implicit surfaces to encapsulate and hide nodes of a cluster and retrieve a more simplified version of the graph without manipulating the graph layout itself. The ASK-GraphView system by Abello et al. [AHK06] is another approach that replaces a group of nodes by a meta-node, similarly to Batagelj et al. [BBD+11]. If the cluster is opened, the underlying nodes are shown, but without any enclosures indicating the clustering and therefore also not altering the graph structure.

If introducing graph aggregation and therefore adding one or multiple abstraction layers over the original data set, a strategy to render and explore the aggregation needs to be considered. Especially if rendering a hierarchical aggregation, different traversal strategies exist to visualize the relations. Elmqvist and Fekete [EF10] summarize different traversal types (see Figure 4.5).

To put the hierarchy traversal strategies into practice, additional interaction techniques have to be introduced to drill-down and roll-up the hierarchy. Common interactions, as described by Elmqvist and Fekete [EF10], are *Coupled Zooming and Drilling*, *Local Aggregation Control* and *Flipping*. *Coupled Zooming and Drilling* is also known as



(a) (b)

Figure 4.4: Hierarchical aggregation of a data set by Archambault et al. [AMA08]. In the graph hierarchy (a) the leaves of the tree represent the original data nodes and the other nodes represent the clusters. The clustered graph (b) is visualized top-down with circular enclosures around the nodes representing the clustering.

semantic zoom or seamless zoom. This type of interaction provides more semantic details the further the user zooms into a visualization. This means that, depending on the zoom level, a deeper level of the hierarchy tree is traversed and rendered. Figure 4.6 shows a visualization of coupled zooming and drilling by Elmqvist and Fekete [EF10]. This also is the interaction we decided to use for our work. The reason is that the zoom interaction itself is well-known to a broad audience. By coupling it to drill-down the aggregation, we aim to make the exploration more naturally.

Local Aggregation Control is needed because the user sometimes wants to expand a cluster in the current context manually, without expanding anything else. Elmqvist and Fekete [EF10] visualize the local aggregation on a hierarchy tree as shown in Figure 4.7. It is a common technique to use basic click interactions on the cluster to expand it and show its subnodes.

Fisheye views are another technique to achieve local aggregation control, which require less input but the user also is more restricted. They were first formally defined for computer interfaces by Furnas [Fur86]. The idea is to show the local area of focus in more detail, but within an abstracted global context. By moving the fisheye, the user can change the focus and the level of detail of local structures in the visualization. If used to explore hierarchical structures, it is called compound-fisheye view [AKY05]. In this case, showing more detail means to show more of the original data and less of the abstract clusters. Therefore, clusters in the focus are expanded, whereas clusters outside the focus are collapsed.

Abello et al. [AKY05] implemented a compound-fisheye view in combination with zooming

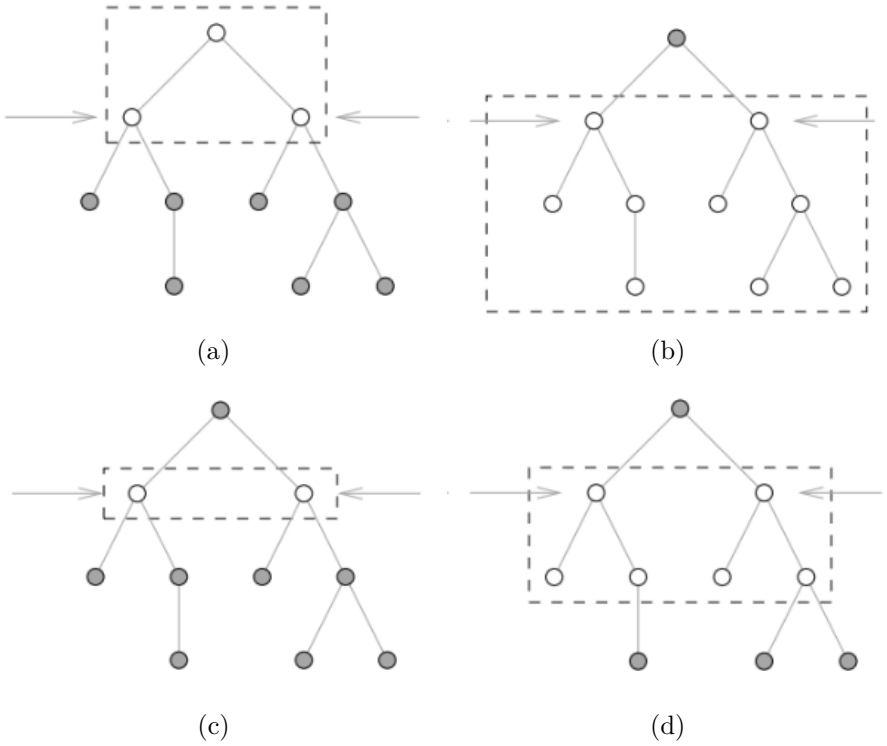


Figure 4.5: Hierarchy traversal strategies by Elmqvist and Fekete [EF10]. (a) depicts the above traversal strategy, (b) the below traversal strategy, (c) the level traversal strategy, and (d) the range traversal strategy.

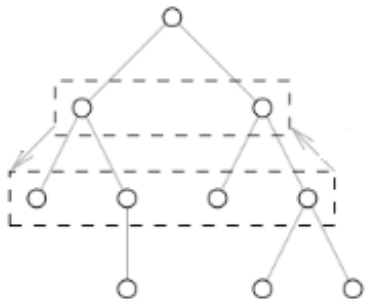


Figure 4.6: Coupled zooming and drilling interaction visualized in the hierarchy tree by Elmqvist and Fekete [EF10]. The arrow on the left side pointing to a lower level of the hierarchy means to zoom into the visualization and show a deeper, more detailed level of the context, whereas the arrow on the right side pointing up indicates to zoom out of the visualization and therefore show a less detailed view.

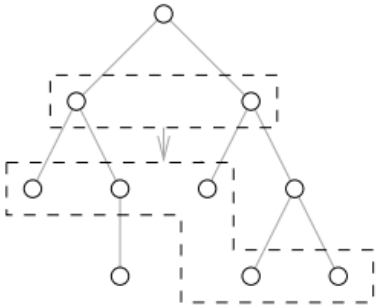


Figure 4.7: Local aggregation interaction visualized in the hierarchy tree by Elmqvist and Fekete [EF10]. It was drilled down to the third level of the hierarchy and the right-most cluster of the aggregation was expanded locally to show its subnodes.

interaction to drill-down large graphs. Hartono et al. [HW16] introduced an approach using a combination of semantic zoom and fisheye view. The fisheye view is used to select the area that should be drilled-down and semantic zoom is then used for the actual drill-down in the hierarchy. Tominski et al. [TAvS06] use the fisheye-view for hierarchies visualized as tree views. They also introduce different types of lenses to apply to a node-link visualization. The local edge lense, for example, hides all edges inside the lense which are not connected to any of the nodes inside the lense to reduce clutter. A drawback of the fisheye view is that the view on the data is distorted.

The term *Flipping* for an interaction technique was first used by Blanch et al. [BL07] in the context of large graph visualization. It was used to describe the interaction of changing from one node to its neighbour and is typically used after zooming into a visualization if only a part of the visualization is rendered. Elmqvist and Fekete [EF10] also used this term and referred to it as geometric pan since this type of interaction only changes the view and not the aggregation. Figure 4.8 shows an abstraction of this interaction indicating that only one level of the hierarchy is affected at a time.

Blanch et al. [BL07] used this interaction for zoomable treemaps in combination with animations to change the focus from the current node to its neighbour. Elmqvist and Fekete [EF10] also implemented it for an example of zoomable treemaps. However, this exploration of neighbours changes the current focus and context. To explore the neighbours of nodes within the current focus, without changing the focus, one possibility is to bring the neighbours into the focus. Tominski et al. [TAvS06] implemented a 'bring neighbor lens'. If applied to one or multiple nodes, their neighbours, which are not already within the current focus, are pulled into the focus. We also make use of this concept in our work.

Edge-Bundling

Edge-bundling aims to improve the readability of large graphs by reducing clutter.

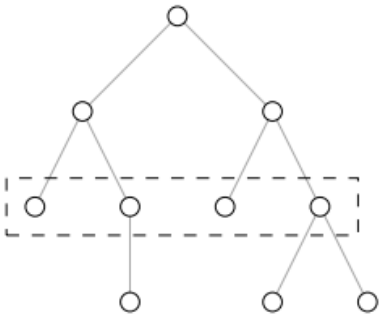


Figure 4.8: Flipping interaction visualized in the hierarchy tree by Elmqvist and Fekete [EF10]. Only one level of the hierarchy is affected at a time. This interaction is used to switch from one node to one of its adjacent neighbours.

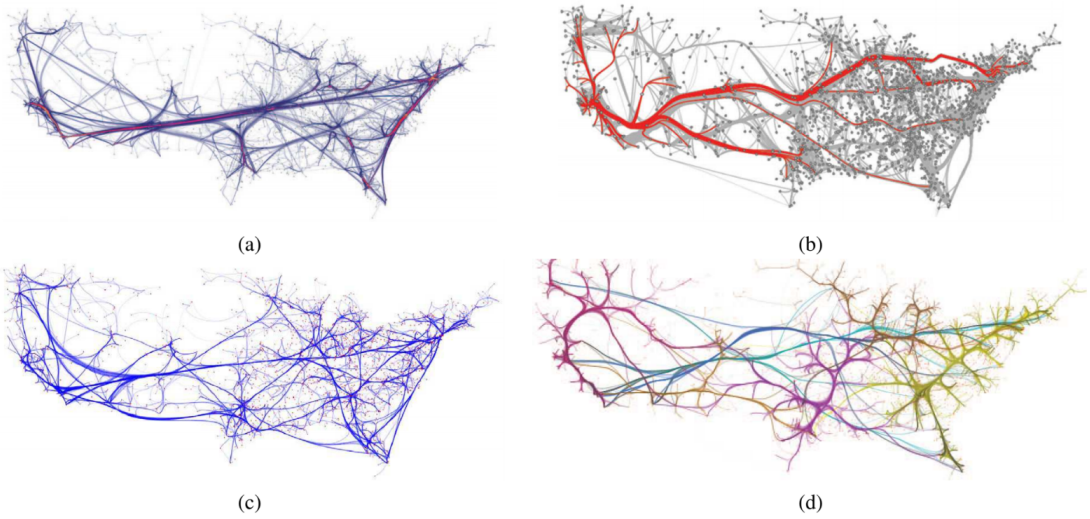


Figure 4.9: Multiple edge-bundling results shown on US migration data, taken from Zhou et al. [ZPYQ13]. (a) force-based edge-bundling by Holten and van Wijk [Hol06], (b) geometry-based approach using triangulation by Cui et al. [CZQ+08], (c) geometry-based approach using quadtrees by Lambert et al. [LBA10], (d) image-based edge-bundling by Ersoy et al. [EHP+11] using a skeleton.

Related edges are clustered into bundles, which are then visualized as curved edges. Edge-bundling only changes the edges and not the nodes. The term edge-bundling was first formally described by Holten et al. [Hol06] and used for the visualization of a hierarchically aggregated graph. In the survey of Zhou et al. [ZPYQ13] edge-bundling is classified into the following three categories:

- **Cost-based Edge-bundling**
  These approaches are based on the minimization of cost functions. The core of the cost functions is to formalize the shape of an edge. Gansner and Koren [GK07], for example, minimize the use of ink for edge bundles in circular layouts. The curves are defined by the two end nodes of the edge as well as two control points that can be moved freely to shape the curve and therefore minimize the ink. Another concept of cost-based approaches is energy minimization, also referred to as force-based edge-bundling. Holten and van Wijk [HvW09] simulate edges as springs with physical forces to attract close edges. A result of this approach can be seen in Figure 4.9a.

- **Geometry-based Edge-bundling**
  Geometry-based approaches use either grids or trees to cluster and form edges. Grid-based approaches take the graph layout and split it into a grid. A grid can be created, as by Qu et al. [QZW07], through a Delaunay triangulation where the vertices of the graph are the control points of the mesh. Cui et al [CZQ+08] generate edges perpendicular to the most important edges and then create a triangulation based on these edges (see Figure 4.9b). Grids can also be based on quadtree partitions [LBA10, LLCM12]. According to the generated grid partitions, the edges are bundled and shaped using control points on the grid edges. Tree-based approaches can be used for hierarchical structures. Holten et al. [Hol06], for example, hierarchically bundle edges at the level of the hierarchy in which the nodes are inside the same cluster.

- **Image-based Edge-bundling**
  The basis of image-based edge-bundling is the computation of edge bundles using common clustering techniques. These edge-bundles are then rendered in a framework using different layout techniques. Ersoy et al. [EHP+11], for example, compute a skeleton based on the generated edge-bundles, which is then used to create an edge-bundling layout where the bundles are routed to the skeleton (see Figure 4.9d).

Although edge-bundling reduces visual clutter, it does not apply to every data set representing a graph. The provided examples in Figure 4.9, showing the US migration, are nice examples for edge-bundling, because the edge length has no meaning, as the underlying geographical map defines the meaning of the node distances. In this thesis, we work with a weighted graph with no predefined, meaningful layout so we need the edge length to represent this weight. Edge-bundling, which alters the edges and therefore their length, is not useful in this case. Additionally, the user study by McGee and Dingliana [MD12] indicates that edge-bundling negatively influences the results of pathfinding tasks and also does not improve the users' performance in terms of finding clusters. The user study by Xu et al. [XRPH12] also shows that curved edges have a bad influence on graph readability, getting worse the stronger the edges are curved. Moreover, they found that users strongly prefer straight edges over curved ones.

4.1.2 Adjacency Matrix

In an adjacency matrix, each entity of the dataset gets a row and a column assigned. There is an entry in the matrix if the entity represented by the corresponding row and the entity of the column are connected. In visualizations of adjacency matrices, the fields of the matrix are usually coloured using a colour scale representing the edge weights. As the example in Figure 4.1b shows, clusters can also be visible in this type of visualization. To achieve this the rows and columns are reordered. Behrisch et al. [BBHR+16] present a survey on algorithms for matrix reordering.

Matrix representations have the advantage that they have a well-defined layout without any overlaps. That is the reason why the user studies by Ghoniem et al. [GFC05] and Keller et al. [KEC06] show that the matrix representation is better for larger and denser graphs. Compared to node-link diagrams it is harder to follow paths for the user and users need the experience to read adjacency matrices.

Also, matrix representations reach a limit in terms of graph size. Therefore, for this type of visualization, approaches to handle larger graphs exist too. ZAME by Elmqvist et al. [EDG+08] is a tool to explore large graphs using a hierarchical aggregation of the dataset and a zoomable adjacency matrix representation. A visualization of a graph at two different zoom levels can be seen in Figure 4.10. Abello et al. [Av04] also implemented a zoomable adjacency matrix. They added hierarchy trees on the rows and columns to support the navigation of the user.

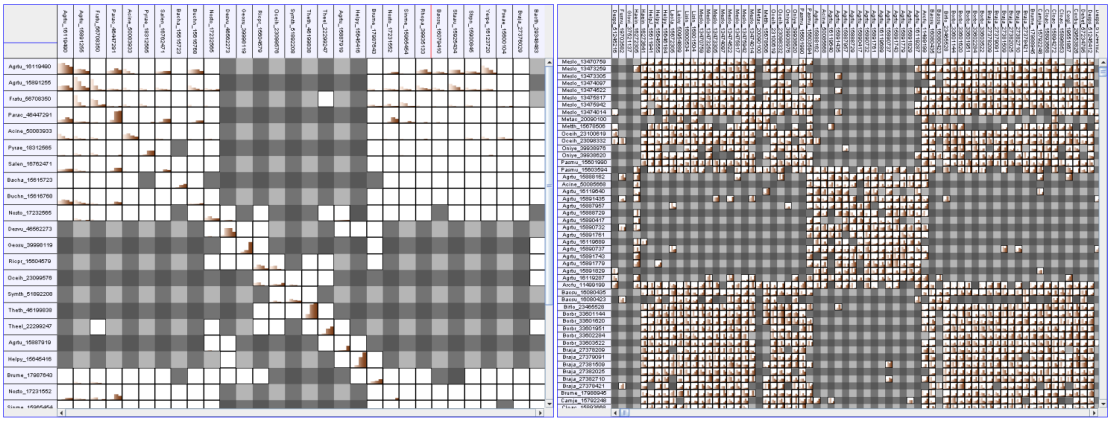


Figure 4.10: ZAME - zoomable adjacency matrix at two different zoom levels by Elmqvist et al. [EDG+08].

As the aim of this work is to provide an intuitive visualization for lay users and emphasize the structural view of the data, we decided not to use matrix representations.

4.1.3 Hybrid Visualization

Combinations of node-link diagrams and adjacency matrices are approaches to combine the advantages of both visualizations. The idea is to use matrix representations for the denser structures in the graph, i.e., clusters, and connect these clusters with links, as in node-link diagrams. Figure 4.11 shows NodeTrix by Henry et al. [HFM07], which uses exactly this idea. Batagelj et al. [BBD+11] visualize clusters not only with matrix representations but using encapsulated node-link diagrams as well (see Figure 4.12). This kind of visualization also does not fit for this thesis as it makes use of adjacency matrices, which are not intuitive, especially for untrained users.

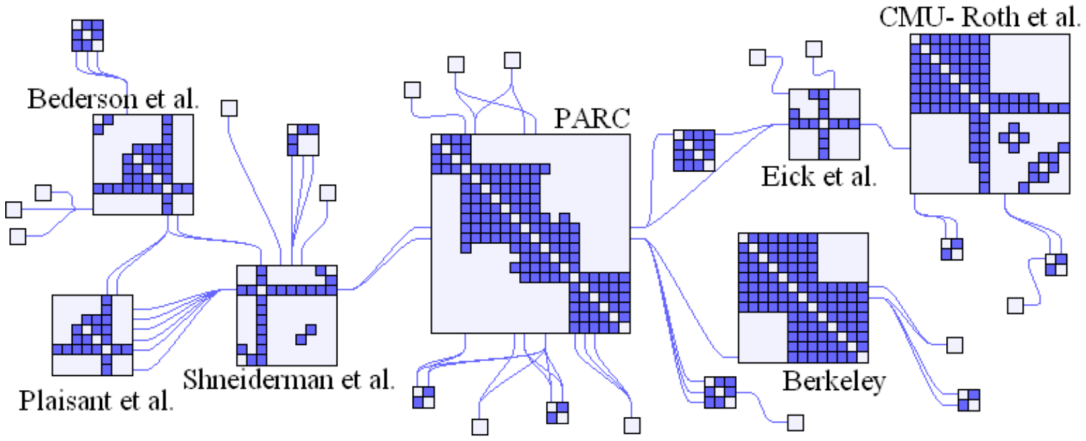


Figure 4.11: NodeTrix by Henry et al. [HFM07]. Clusters are visualized using adjacency matrices and are handled as nodes in a node-link diagram, which are connected by links.

4.2 Bipartite Graph Visualization

Although the previous subsections handled general graphs, many aspects can be applied to bipartite graphs as well. Some of these aspects will be discussed in the following in terms of their applicability to bipartite graphs. The goal of bipartite visualizations is to support the understanding of the relations between the entities of the involved two sets.

For bipartite graphs increasing the size of the graph is an issue as well. The basic concept of graph aggregation is the same as for general graphs. Additionally, biclustering algorithms exist that cluster the rows and the columns of a biadjacency matrix at the same time and can be used for bipartite datasets. There exist multiple surveys of biclustering algorithms [MO04, TSS05, PZT+06, VMB+10, BKC10, PC17]. Biclustering algorithms are widely researched due to their importance in the analysis of gene expression data. In this context, it makes sense that genes can be in more than one cluster at the same time. However, for graph aggregation and the visualization of it, overlapping sets are a big problem due to the number of possible combinations. Moreover, we want to generate

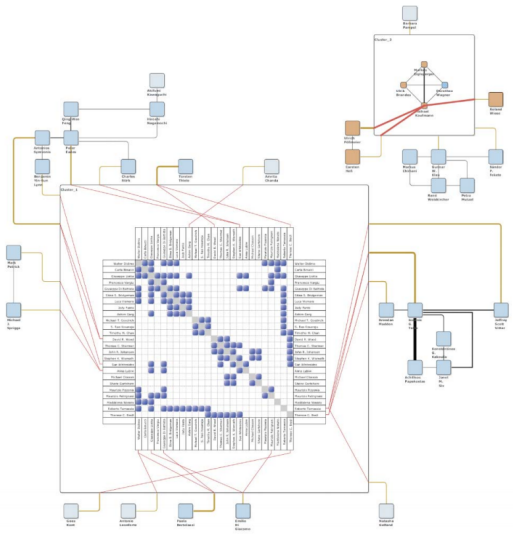


Figure 4.12: Hybrid Visualization by Batagelj et al. [BBD+11] using encapsulated matrix representations (cluster in the middle) as well as node-link diagrams (top right cluster) for clusters.

path-preserving hierarchies. To acheive this, it is necessary to assign a node to exactly one cluster and work with distinct clusters. As already mentioned in the previous section, we use agglomerative clustering instead of biclustering. Edge-bundling is used for bipartite graph visualizations as well and can be used in the same ways as described for general graphs.

In the next paragraphs, we will discuss approaches that make use of the previously described visualization techniques of general graphs and see how they were adapted for bipartite graphs. Due to the characteristics of the data, it is natural to split the entities in the visual space as well.

Pezzotti et al. [PFH+18] implemented a multi-scale node-link visualization using a parallel layout to split the sets, as well as different colours to emphasize the sets, and edge-bundling. They use a force-based layout to position the nodes. Figure 4.13 shows a visualization of their approach. The vertical position of the nodes in the parallel layout is set according to the force-based layout and for each set, the force-based layout of the set nodes is shown on the respective side of the parallel node-link diagram. The force-based layouts are visualized using dots. The entities are drawn as different-sized transparent dots. The size indicates the number of vertices the cluster contains. The transparency is used to show the overlap, more precisely, to indicate how connected the nodes are. They implemented a top-down visualization that can be drilled-down by selecting a set of clusters (brushing). The brushed clusters are then shown in more detail. One drawback of this visualization is that the nodes are highly overlapping and therefore it suffers from readability. It is hard to find paths between nodes in this overlapping layout. The force

layouts on the side offer a structural view on the data in terms of group membership or how many clusters there are. But the connections of these clusters or inside these clusters cannot be seen easily.

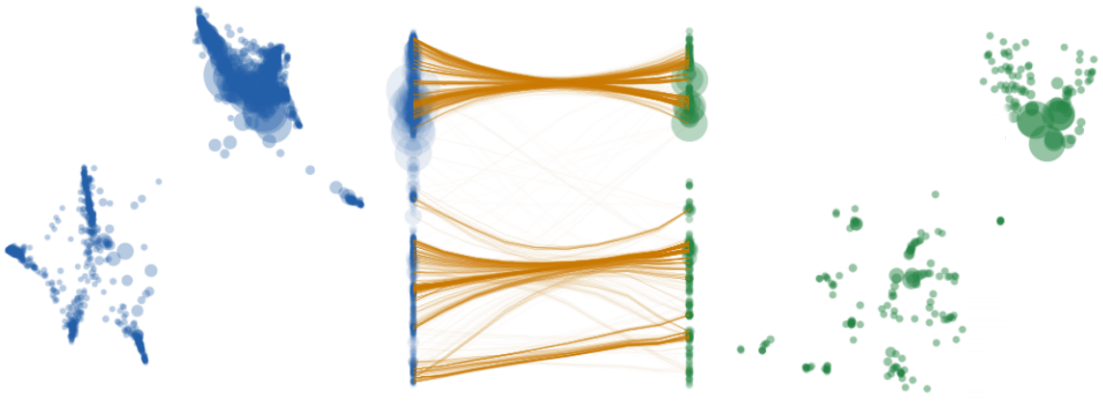


Figure 4.13: Multi-scale node-link visualization of a bipartite graph by Pezzotti et al. [PFH+18]. The sets and their entities are positioned in a parallel fashion according to an underlying force-based layout. The nodes of each set are additionally visualized as dots based on their position in the force-based layout.

Another approach using node-link diagrams are anchored maps by Misue [Mis06]. Nodes of one set are anchored to a defined position and the nodes of the other set are arranged around the anchor nodes according to their relations. Figure 4.14 shows an anchored map by Misue [Mis06]. The anchor nodes are usually positioned in a radial layout. The problem of this approach is that it requires an unbalanced dataset where one of the sets is smaller than the other one. As we do not want to assume this kind of property from the dataset, the approach is not used for this thesis.

BicOverlapper is a tool by Santamaria et al. [STQ08] that is used to analyse, explore and compare biclustering algorithms. They use a force-based node-link visualization where the entities of one set are drawn as rectangles and those of the other set as dots. The nodes of a bicluster are overlayed with an opaque layer. Since biclusters are not distinct, the problem of overlapping set visualization and the possible number of combinations can be seen very well in their visualization.

Matrix representations are not common for visualizing bipartite data sets. Instead of an adjacency matrix, a biadjacency matrix is used. Biadjacency matrices and their visualizations are typically used in the research area of biclustering algorithms. However, this area has its focus on the biclustering algorithms and the corresponding cluster analysis and not the data exploration itself. Barkow et al. [BBP+06], for example, implemented a toolbox that allows the user to select from multiple biclustering algorithms. The

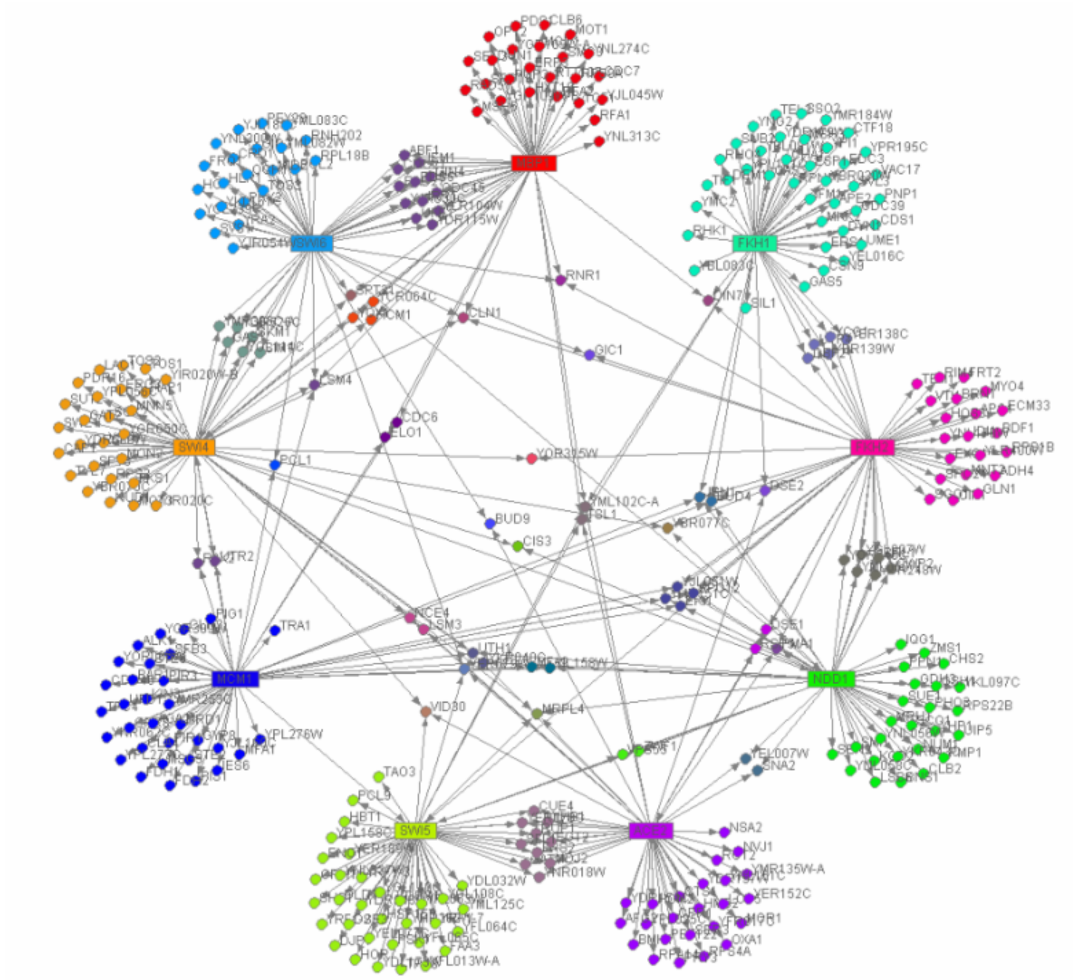


Figure 4.14: Anchored Maps by Misue [Mis06] where nodes of one set are anchored in a radial layout and rendered as rectangles and the nodes of the other set are placed according to their relations to the anchor nodes and drawn as dots.

biadjacency matrix of the data is visualized as heat map and rows and columns are reordered such that clusters appear in the top left of the map.

Hybrid visualization, as discussed in the context of general graph visualization, are also typically used in the context of biclustering. Examples of tools used for the analysis of biclustering algorithms, which are very similar to NodeTrix [HFM07], are Furby by Streit et al. [SGG+14], the approaches by Xu et al. [XCQS16], and Fiaux et al. [FSB+13].

For bipartite sets often other visualizations than the previously mentioned ones are used. Figure 4.15, for example, shows one of the most famous Sankey diagrams [KM13]. It depicts a map of Napoleon's Russian Campaign of 1812 made by Charles Minard. It is a

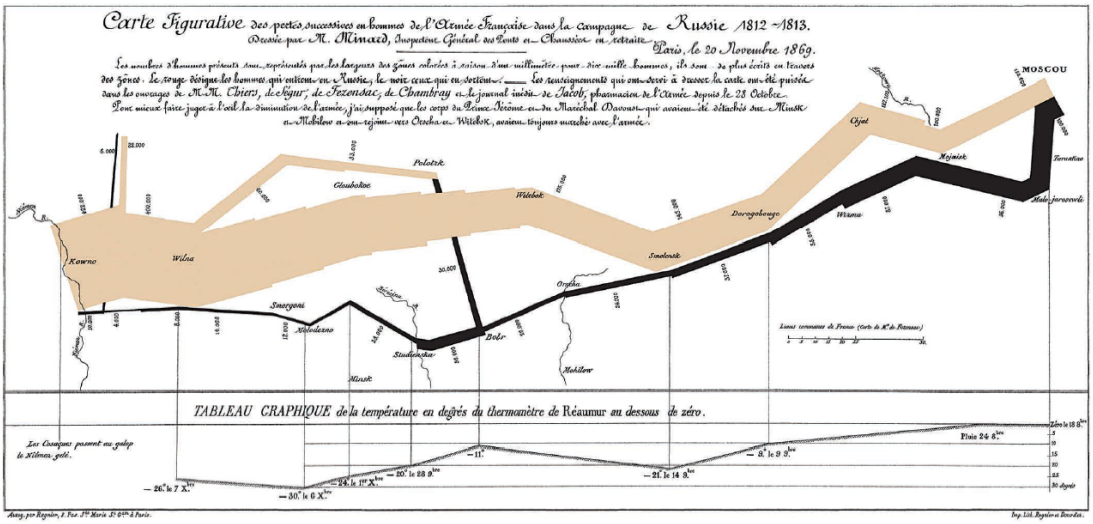


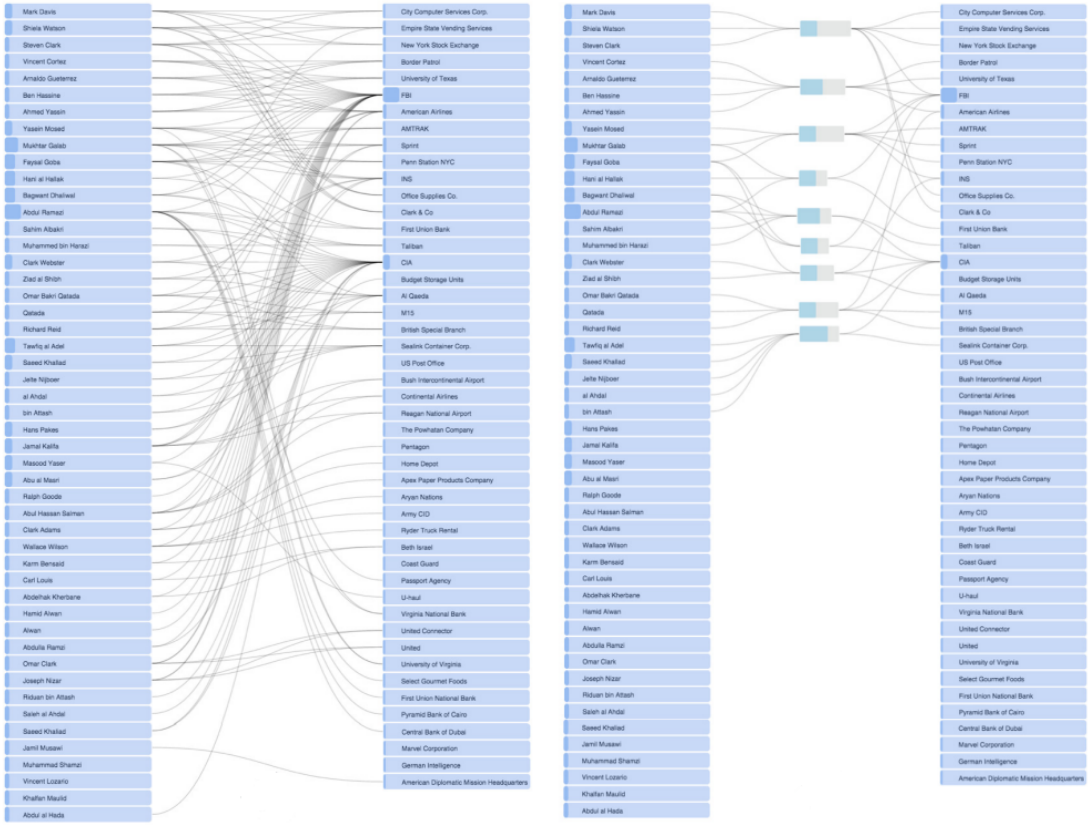
Figure 4.15: Sankey Diagram by Charles Minard showing a map of Napoleon's Russian Campaign of 1812. [KM13]

flow diagram showing the loss of men of Napoleon's army during their march. The width of the line represents the number of men.

The connections between two sets can also be interpreted as a flow from one set to the other one. Not only Sankey diagrams can be reinterpreted in this manner. Also, parallel coordinates can be used such that there are only two axes that represent the two sets. This approach is often also called parallel sets [BKH05]. If the axes are replaced by lists, the term parallel list or list view is used. These kind of visualizations are very similar since they have a parallel layout in common, where entities are connected by edges.

Netflower by Stoiber et al. [SRG+19] is an approach based on parallel sets for dynamic bipartite graphs. The entities are shown on the set axis as a bar where the height of the bar represents the data value. The outgoing edges show the connections to entities of the other set with an edge width according to the data of this relation. Each entity has a bar chart on its side to show the value changes of the respective entity over time. This approach only shows a few entities, i.e., the ten largest entities, at a time and the context of the entities within the whole dataset is lost.

Jigsaw by Stasko et al. [SGLS07] describes the implementation of a tool for data exploration in the area of document analysis, which uses parallel lists. A list view is used to show connections between user-defined sets of entities. Additionally, the tool offers a graph view to show the connections of documents and entities in a node-link diagram. Ghani et al. [GKL+13] introduced parallel node-link bands, which is a visualization similar to Jigsaw and used for social network analysis. Also for parallel lists, scalability is a big issue. Sun et al. [SMNR16] also use parallel lists for the visualization of two sets.



(a) (b)

Figure 4.16: BiSet by Sun et al. [SMNR16]. (a) depicts parallel lists without edge-bundling. (b) shows the same dataset with edge-bundling and hides single links which do not belong to any bicluster.

They try to reduce clutter and improve readability using edge bundling. Biclustering is used to detect groups in the dataset. The edges inside a group are formed into edge bundles. Moreover, single edges, which do not belong to any cluster, are hidden from the visualization. Figure 4.16 gives an example of the parallel lists by Sun et al. [SMNR16] with and without edge-bundling.

BiCFlows by Steinböck et al. [SGW18] is based on Sankey diagrams as well (see Figure 4.17). Their approach handles larger datasets with a multi-scale approach. The data is recursively biclustered to retrieve a hierarchy tree. The data is rendered top-down such that the whole dataset can be seen at once. To drill-down in the hierarchy tree, the user clicks on a cluster, which is then opened to show the subnodes and their connections. The drawback of this visualization is that it is not for non-expert users. Their user study showed that users are encouraged to explore the dataset in more depth compared to an example without aggregation, but that the users need time to understand the

visualization and aggregation. The approach in this thesis is based on BiCFlows. With our approach, we try to introduce a more intuitive visualization using node-link diagrams instead.

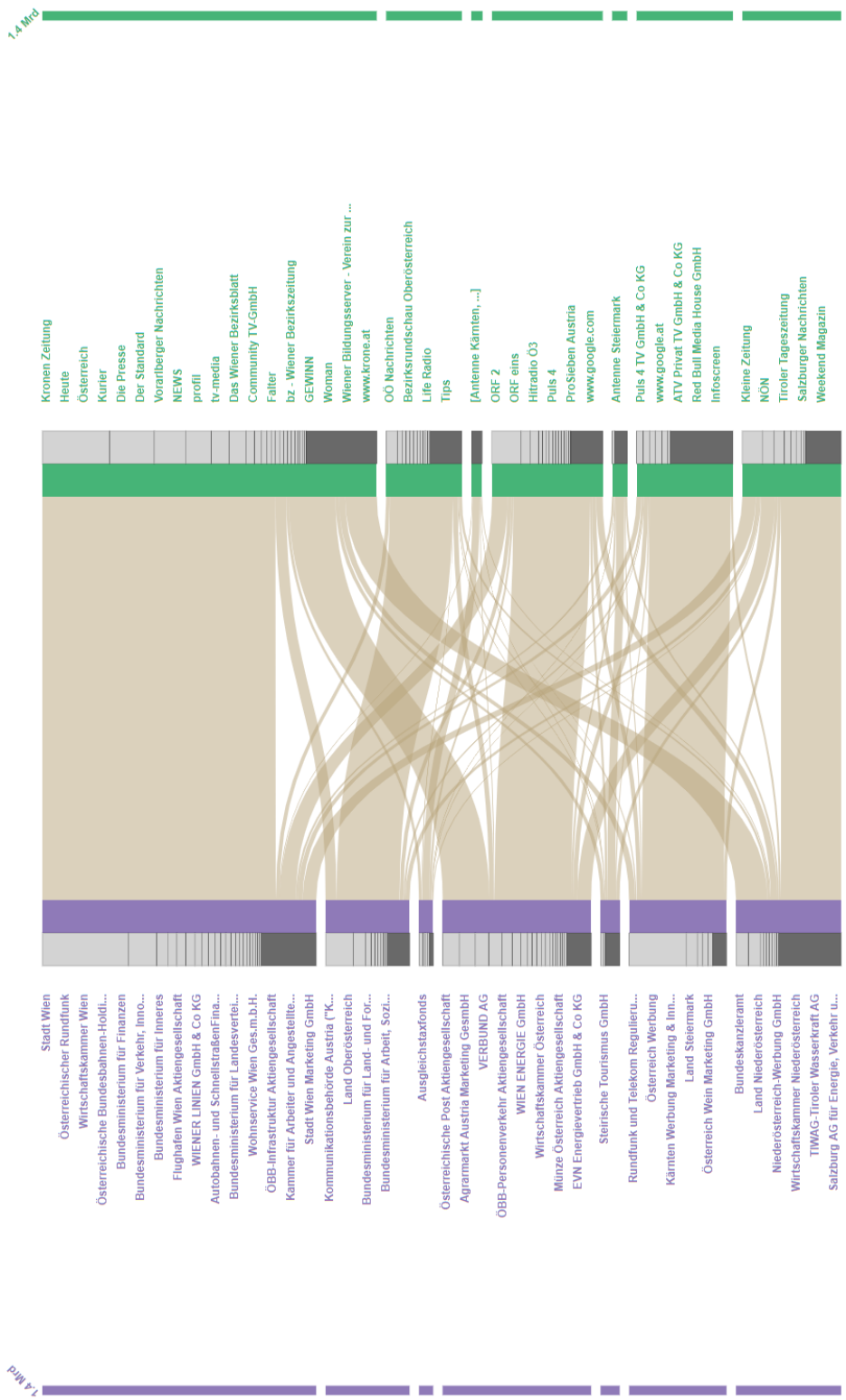


Figure 4.17: BiCFlows by Steinböck et al. [SGW18] is a mutli-level approach based on Sankey diagrams.

CHAPTER 5

Visualization and Interaction Design

In Chapter 3 we showed that graphs beyond 10000 edges cannot be visualized as classic node-link diagrams on the web on average consumer hardware, without reaching the hardwares limitations or visual clutter. In this chapter, we will introduce Firework Plots, a visualization concept, which aims to handle these scalability issues. Moreover, we will describe which interaction techniques are used to support the exploration and describe user interface.

5.1 Firework Plots

In Chapter 1 we described the requirements our visualization should fulfil. Achieving many of these requirements results in the concept of Firework Plots, which is a combination of multiple techniques that will be described in the following. We use node-link diagrams as a basis for visualization as it has been proven to be intuitive for inexperienced users in multiple user studies (see Chapter 4). To overcome scalability issues, we use hierarchical aggregation. As this approach offers views with different levels of detail, this also allows us to provide an overview of the data. Moreover, we describe two interaction techniques, i.e., seamless zoom and context pull, as part of our visualization concept, which aim to support in-depth exploration.

5.1.1 Node-Link Diagram

As described in the related work in Chapter 4, node-link diagrams are the most intuitive choice, as long as the graph does not become too big or too dense. We decided to use node-link diagrams even though we are focusing on the visualization of large graphs. To handle large graphs, we used different concepts, which will be covered in the next

subsections. Figure 5.1d shows the visualization of a small test graph, which represents a small subset of the Media Transparency Dataset. The entities of the underlying dataset are represented as nodes, which are drawn as filled circles. The colour of the circle represents to which of the two sets the entity belongs to. Nodes of the first set are coloured in light blue and nodes of the second set are coloured in dark blue. The size of a circle encodes the weight, which is taken from the underlying dataset. As the dataset specifies relations (edges) and their weights, the size of a node is calculated using its incident edges. Let $V_1 = \{u_1, ..., u_n\}$, $V_2 = \{v_1, ..., v_m\}$ and s_{u_i}, s_{v_j} be the size of the corresponding nodes, then the size s_{u_i} of a nodes u_i of the first set V_1 is the sum of the weights w_{ik} of all relations $e_{ik} = (u_i, v_k) \in E$:

$$s_{u_i} = \sum_{k=1..m} w_{ik}. \tag{5.1}$$

The same way, the size s_{v_j} of a node v_j from the second set V_2 is calculated as:

$$s_{v_j} = \sum_{l=1..n} w_{lj}. \tag{5.2}$$

These sizes are then normalized and interpolated between a fixed minimum and maximum circle area. The corresponding nodes in the node-link diagram are then rendered with the calculated circle area. The width of the line shows the weight of a relation. In the same way, as for the nodes, the link weights are normalized and linearly interpolated between a minimum and maximum width.

5.1.2 Hierarchical Aggregation

The main concept we use to handle large graphs is the hierarchical aggregation of the dataset. We decided to use a topology-based clustering algorithm to obtain a graph hierarchy. With it we wanted to emphasize the structural view of the data. Moreover, we use an implementation of agglomerative clustering based on the publication of Newman et al. [New04]. Agglomerative clustering starts with every node defining a cluster. Based on a quality function, clusters are merged. Their quality function to calculate if a separation is meaningful or not is defined as:

$$Q = \sum_i (w_{ij} - a_i^2) \tag{5.3}$$

where w_{ij} is the weight of the edges e_{ij} that connect the nodes of cluster i and cluster j and $a_i = \sum_j w_{ij}$. If $Q = 0$ this means that a cluster does not include more edges than it would have if edges were randomly assigned. This process of agglomerative clustering guarantees that nodes which are in the same cluster are connected. Besides, this is important to secure a path-preserving hierarchy. Based on the quality calculations the implementation also determines the best number of clusters. From the initial calculations, it is not possible to determine the best number of clusters for a subgraph. Therefore, if a cluster exceeds a certain size, the subgraph that is defined by this cluster is clustered

again to determine the best groupings and so on. Figure 5.1 shows the graph hierarchy of our test graph as well as the visualizations of each level of the graph hierarchy. In the node-link diagram, clusters are rendered as grey circles. In the previous subsection the size of a data node was described to be calculated as the sum of the weights of the outgoing links. In the same manner, the size of a cluster is based on the sum of the weights of the outgoing links. Since a cluster can also aggregate complete links if both endpoints are assigned to this cluster, these links should also be considered for the cluster size. Therefore, the cluster size is calculate as the sum of the weights of the outgoing links out_c and the weights of the inner links in_c. The outgoing links are all links which are incident to exactly one data node that was assigned to this cluster. Let $V_c = V_{1c} \cup V_{2c}$ be the set of nodes that were assigned to the cluster c according to their set membership. Then the weight of the outgoing links of this cluster can be calculated separately for each set:

$$out_{1c} = \sum_{i=1..|V_{1c}|} \sum_{j=1..|V2 \setminus V_{2c}|} w_{ij} \tag{5.4}$$

and

$$out_{2c} = \sum_{i=1..|V_{2c}|} \sum_{j=1..|V1 \setminus V_{1c}|} w_{ij}. \tag{5.5}$$

The sum of the weight of links whose endpoints are both assigned to the same cluster is calculated as follows:

$$in_c = \sum_{i=1..|V_{1c}|} \sum_{j=1..|V_{2c}|} w_{ij}. \tag{5.6}$$

The size s_c of a cluster node c is then calculated as the sum of the outgoing edges as well as the aggregated, inner, edges:

$$s_c = out_{1c} + out_{2c} + in_c. \tag{5.7}$$

5.1.3 Seamless Zoom

We use coupled zooming and drilling to drill-down the graph-hierarchy. This interaction technique is also known as semantic zoom or seamless zoom. This means by zooming into the visualization, more details are shown. In terms of this thesis, more details correspond to a finer clustering of the data, i.e. showing the next level of the graph hierarchy. The zooming interaction itself already is a common interaction technique. By coupling it with the drilling of the hierarchy we aim to receive an understandable and self-explanatory interaction technique. In the following we refer to the *next* level or *down* if we mean the neighbouring level of the hierarchy with more details and to the *previous* level or *up* if we mean the level with less detail. We implemented a classic zoom using the mouse wheel. We implement the node-link diagram in a 3D scene and set the distance of the camera to zoom in our out of the visualization. To obtain seamless zoom, we need to specify at which distance the level switches are done. We specify an interval $[50, d_G]$

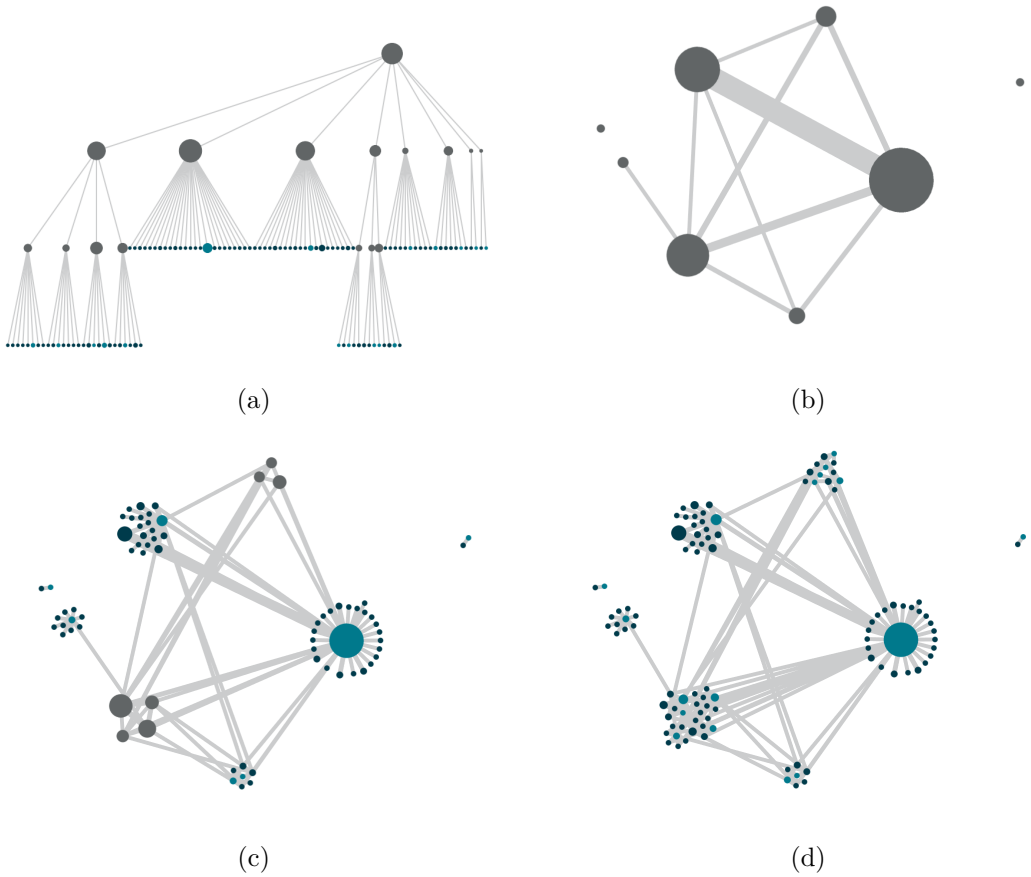
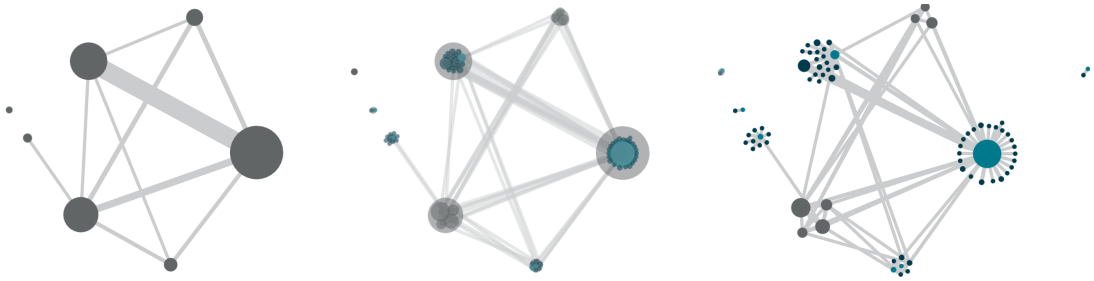


Figure 5.1: (a) shows the graph hierarchy resulting from recursive clustering. (b), (c), and (d) illustrate parts of the hierarchy using a level traversal strategy where (b) depicts the first level of the graph and (c) and (d) show the second and third level. The last level shows all data nodes without any cluster nodes or links. Cluster nodes are coloured in grey, nodes of the first set in light blue, and nodes of the second set in dark blue.

where d_G is the distance of the camera from the graph such that the graph fits exactly inside the frustum of the camera (see Chapter 6). The level switches inside this interval are calculated on a logarithmic scale as recommended by Furnas et al. [FB95]. We found 50 to be a good minimum distance based on the size of the scene elements and the amount of detail this distance reveals. To switch between the scene elements of two levels, animations to visualize the transitions are used. This allows a smooth transition between the levels, which aims to help the user to follow the hierarchical structure of nodes. As soon as the animation is started, the zooming interaction is blocked. After the animation is done, which takes ~ 1.5 seconds, the user can continue to zoom in. The animation itself consists of two parts. The opacity blending of the elements is one part. For each node and link one of three cases applies for the opacity blending. There can be cluster

nodes and links in the current level, which have to be faded out during the animation. The subnodes and links of these cluster nodes need to be faded in. Moreover, there also can be nodes that are visible through multiple levels and are not influenced by opacity blending during the animation. The other part of our animation is the positioning of the nodes. For the animation, the subnodes of a cluster are positioned at the cluster position of the previous level. From this position, the subnodes move to their calculated positions of the new level. For the registration of the levels to each other see Subsection 5.1.4. Figure 5.2 shows images at discrete steps during the animation. This animation looks like an exploding firework hence the name Firework Plots.



(a) (b) (c)

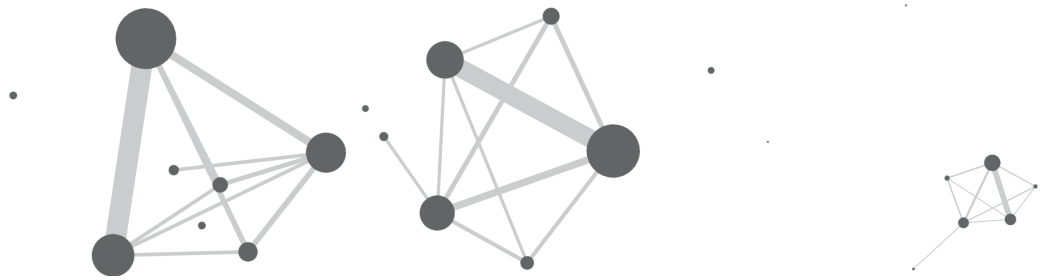
Figure 5.2: Images of the animation between two levels. (a) shows the first level and (c) shows the second level. (b) shows the visualization in the middle of the animation.

5.1.4 Layered Force-Directed Layout

As mentioned in the related work (see Chapter 4), force-based layouts are the most intuitive layout techniques and come the closest to human-made layouts. From the top of the graph hierarchy to the bottom, we calculate a constrained force-directed layout for each level. However, we have to consider how the levels are registered with each other. We calculate, beginning from the top, one level at a time. After the simulation of a level is done, the positions of the nodes at this level are fixed. For a fixed cluster, its subnodes will be constrained to the clusters position at the next level. This keeps the subnodes spatially close to the parent clusters position. This registration also helps to preserve understandability of the animation between the levels, as the subnodes do not move over the whole visualization. As this layered computation is very time consuming, we decided to let the computation be done in worker threads such that the main thread serving the web application is not blocked and the calculations are done in the background. Chapter 6 provides further details about the implementation.

For the simulation of the graph layout at each level, we make use of several different, predefined, force constraints. According to its name, the **charge** force of a force layout simulates an electrical charge that causes nodes to repel each other. More generally this leads to a layout that is expanding in size. We set charge to be stronger at early levels such that there is more space for the subnodes in the lower levels. In Figure 5.3, the

results of different charge settings on the test graph can be seen. It is desirable that the smaller connected components are not overlapping with the larger component as can be seen in Figure 5.3a. Overlapping might lead to unnecessary clutter and at more detailed levels it might be hard to distinguish, which nodes belong to which connected component. Moreover, we do not want the charge settings to be so high that the distance between the connected components is getting too large as in Figure 5.3c.



(a) (b) (c)

Figure 5.3: Three layouts using different charge settings. Low charge in (a) to large charge in (c).

Moreover, we use **collision avoidance** to avoid overlapping nodes. This ensures that all nodes are visible and that there is a clear layout that minimizes misinterpretation due to overlaps.

Attractive forces are used to define positions to which nodes should be pulled. In our case, we use these forces in the levels apart from the first one. For each subnode we set these forces to its parent's position. This way we can ensure that the subnodes are close to the parent nodes' position and our firework animations are easier to follow even when zooming in.

As a last type of constraint, we make use of the **link distances**. This constraint specifies the distance between adjacent nodes. We make use of the edge weights defined in the dataset. A low weight w_{ij} in the dataset describes a weak relation whereas a high weight corresponds to a strong relation. To calculate the link distances dl_{ij}, the weights w_{ij} in the dataset are normalized and written as $w_{norm_{ij}}$. Then the inverse of the normalized weights are calculated and used as link distances:

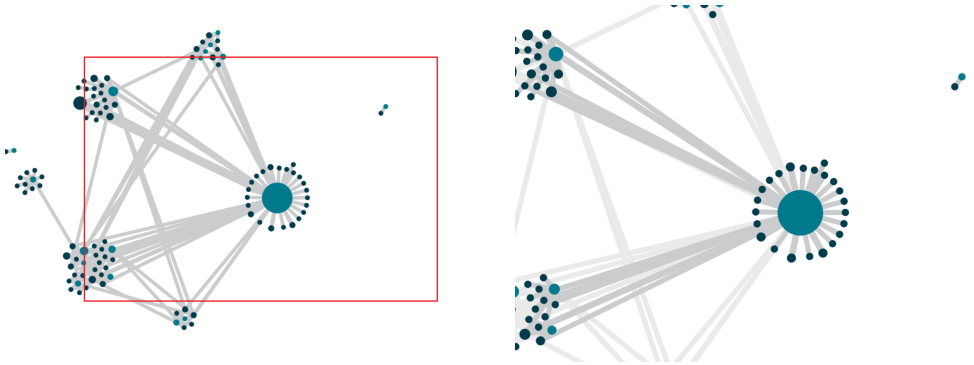
$$dl_{ij} = 1 - w_{norm_{ij}} \tag{5.8}$$

5.1.5 Visibility Management

In Chapter 3 it has already been shown that rendering graphs smoothly cannot be guaranteed anymore above a certain size. Visibility management does not just aim to reduce clutter, but also to improve performance. We are using hierarchical aggregation in combination with seamless zooming. This also means that the image either shows a lot of data abstracted using clustering or it shows data in detail, but only a small part of it.

To improve performance it needs to be considered that, in the detailed image, frustum culling should be used in a meaningful way. We aim to create an assignment of nodes to meshes, such that we have meshes with small bounding boxes. Here it needs to be considered that not too many meshes are created as this decreases the rendering performance. We decided to create these meshes for every cluster of the first level of the hierarchy. All child nodes and links of a cluster are assigned to the same meshes. Due to our graph layout approach, we can assume that all child nodes of a cluster are spatially close. This approach leads to large meshes with small bounding boxes. This helps to make use of frustum culling during the rendering process and to improve the performance. Since this has a lot to do with the implementation of the concept itself, this is described in more detail in Chapter 6.

To improve the perceptual scalability issue, i.e., reducing visual clutter, we are changing the appearance of links based on the position of the endpoints of the links. When zoomed in to a detailed view of the graph, many edges are visible. These edges can be categorized into three cases. To show these cases we use Figure 5.4, where Figure 5.4a shows the whole test graph without aggregation as well as a red rectangle depicting a zoomed-in part and Figure 5.4b shows exactly this zoomed part. The first case is that an edge



(a) (b)

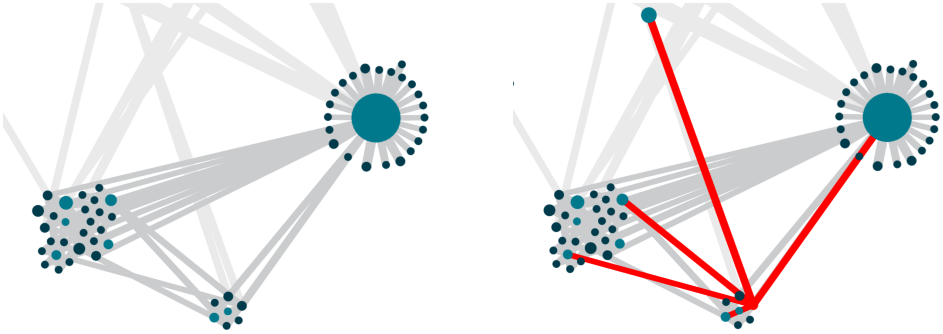
Figure 5.4: (a) shows the whole test graph without aggregation with a red rectangle that depicts the part which is zoomed to at (b). Links whose endpoints are not within the rectangle are not rendered (see the links between the cluster at the top and the one at the bottom left). Links, where only one endpoint is rendered, are rendered in a lighter colour and are positioned in the background (e.g. links from the large light blue entity to nodes at the bottom, outside of the rectangle).

and both of its endpoints are visible. In this case, the link is rendered in the foreground with the specified link colour. In Figure 5.4, this case applies to all links between nodes within the red rectangle. These edges provide the most information, as everything that is directly related to it is visible. The second case covers edges, where only one endpoint is visible. These edges are positioned in the background and rendered using a less prominent colour. In Figure 5.4 examples of such edges can be found between nodes of the large

light blue entity at the centre of the red rectangle and, for example, the nodes at the bottom that are outside of the red bounding box. The last case describes edges that are passing the zoomed-in view but where none of the endpoints is visible. The user does not see any relation to these edges and they only add visual clutter. Therefore, when zoomed-in, we do not render such edges at all. The edges which connect nodes at the top, outside of the rectangle, to those at the bottom left, outside the rectangle, belong to this category. In Figure 5.4b these edges are hidden.

5.1.6 Context Pull

In zoomed-in views as in Figure 5.5, we can see that many links are cut off as they leave the view. If a certain node is of interest to a user, also its neighbours can be of interest, but might not be within the current view. This is the reason we implemented an interaction to *pull* the neighbours of a selected node into the current context. This can be be seen in Figure 5.5. A node is selected by clicking on it. All its direct neighbours are then pulled into the view along their link. Outgoing links of pulled neighbours, apart from those connected to the selected node, are disregarded to reduce clutter. As long as a node is selected, the neighbours are pulled into the view even during zooming or panning. Moreover, when a node is selected, on-demand labelling, as described in the next subsections, only works on incident links and adjacent nodes of the selected node. Additionally, information of the selected node is shown in the sidebar. This information contains the attributes, which are defined in the dataset, i.e. which set the node belongs to and its relations and their respective attributes.



(a) (b)

Figure 5.5: (a) shows a zoomed part of the test graph. In (b), one neighbour at the top of the selected (clicked) node is pulled into the current view.

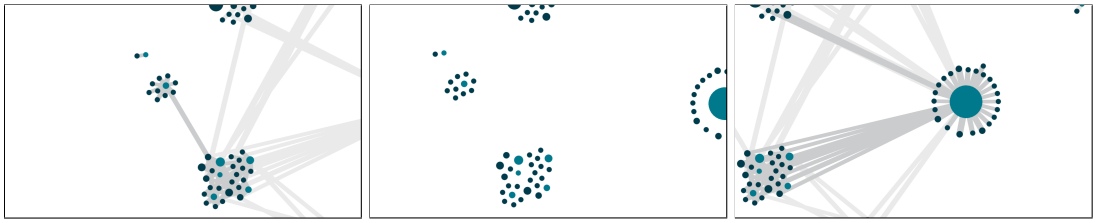
5.2 Interactive Exploration

For the interactive exploration of datasets, different interaction techniques are suitable and needed to support different types of graph analysis tasks. In the following subsections,

we will describe the interaction techniques we chose to implement to support in-depth exploration of the underlying dataset. Seamless zoom and context pull were already described as part of the Firework Plots and will not be handled again in the next paragraphs although they also belong to interaction techniques.

5.2.1 Pan

We implemented a classic pan interaction to move the context within a single hierarchy level. Figure 5.6 shows an example of this interaction where the context before panning, during panning and after panning is shown. The pan interaction is done by pressing the left mouse button and moving the mouse and context. As long as the mouse button is pressed, the links of the visualization are hidden, as it reduces the clutter and improves the performance of the interaction.



(a) (b) (c)

Figure 5.6: Images showing the pan interaction. (a) shows the visualization before panning. (b) shows the visualization during the interaction where the links are not rendered and (c) shows the end position after panning.

5.2.2 Labelling

To get more information about the nodes and edges, we use details-on-demand labels. If a node or link is hovered, we show a label with information about the hovered item. As can be seen in Figure 5.7, during hovering a node the name of the entity represented by the node is shown as well as the sum of weights of those edges connected to this node. Moreover, the node itself and all connected links are highlighted. When hovering a cluster node, the largest node of each set, with respect to its size calculated as in Equation 5.1, within this cluster is shown as well as the cluster size. Also links can be hovered. Then a label, showing the weight of the relation represented by this link, is drawn.

5.2.3 Highlighting

The highlighting on hover was already mentioned before. Moreover, we enable the user to disable rendering links. When a node is hovered in this mode, we also render the incident links to investigate the relations and explore adjacent nodes. An example of this highlighting can be seen in Figure 5.8. The same example but where the links are rendered is given in Figure 5.7.

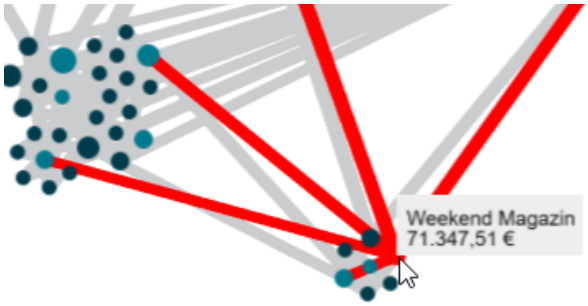


Figure 5.7: Labelling and highlighting of a node when it is hovered.

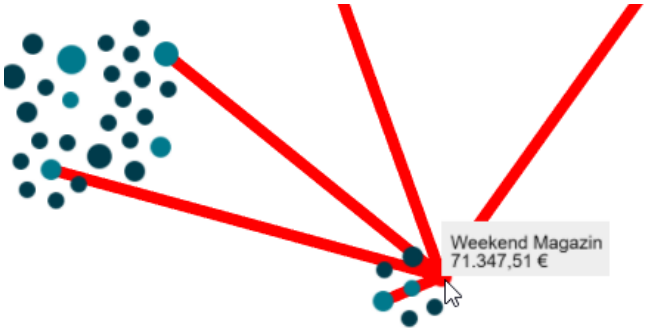
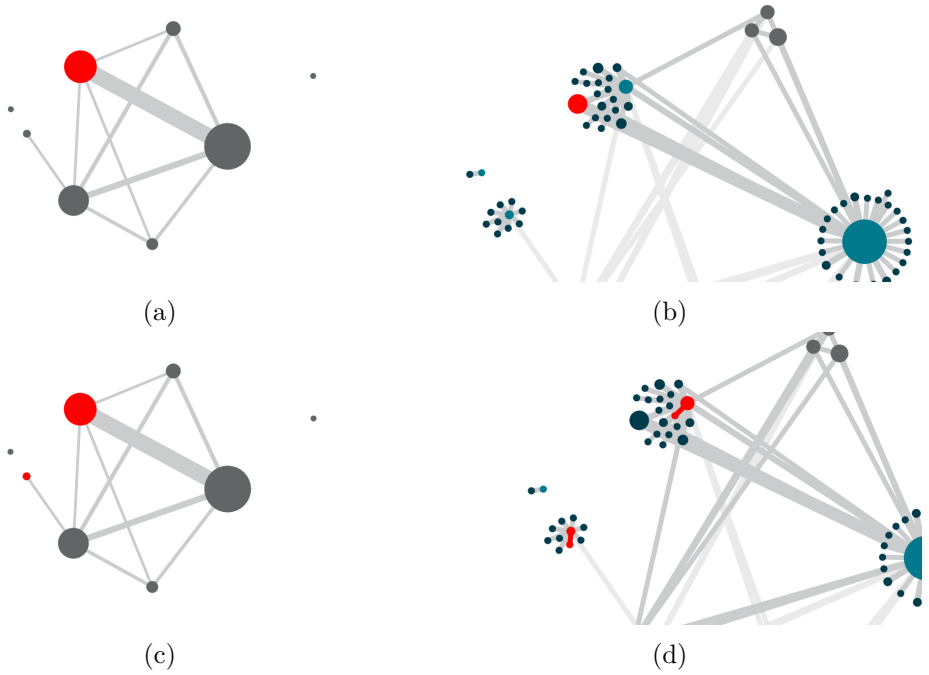


Figure 5.8: Labelling and highlighting of a node if it is hovered, but rendering links is disabled apart from the highlighted ones.

Beside highlighting on hover, we also implemented attribute-based highlighting of graph elements. This supports the in-depth exploration of the dataset. It provides the possibility to highlight elements of the graph based on an attribute, but to still see the elements within the whole context. If a highlighting is applied, it is propagated through the hierarchy. This means the highlight persists during zooming and switching hierarchy levels. This is achieved by highlighting cluster nodes and cluster links if at least one of the underlying data entities is affected by the highlighting.

Figure 5.9 shows two examples of highlighting. Figure 5.9a and 5.9b show a highlighting based on the node label *ORF eins* at two different levels. In Figure 5.9a the first level of the graph hierarchy is shown where the parent cluster of the node with the label *ORF eins* is highlighted. At the next level in Figure 5.9b this entity is already visible and highlighted as well. Figure 5.9c and 5.9d depict a highlighting of entries in the dataset based on §4 of the Austrian *Medienkooperations- und -förderungs-Transparenzgesetz* [Med]. Figure 5.9c shows the first level of the graph hierarchy. In contrast to the first highlighting example, this attribute is a link attribute, not a node attribute. In this case, a cluster is highlighted



(a) (b)

(c) (d)

Figure 5.9: Examples of attribute-based highlighting of the test graph showing a subset of the Media Transparency Dataset. (a) and (b) show a highlighting of the node *ORF eins* and the propagation through different levels. (c) and (d) show the highlighting of relations which are based on §4 of the Austrian *Medienkooperations- und -förderungs-Transparenzgesetz* [Med].

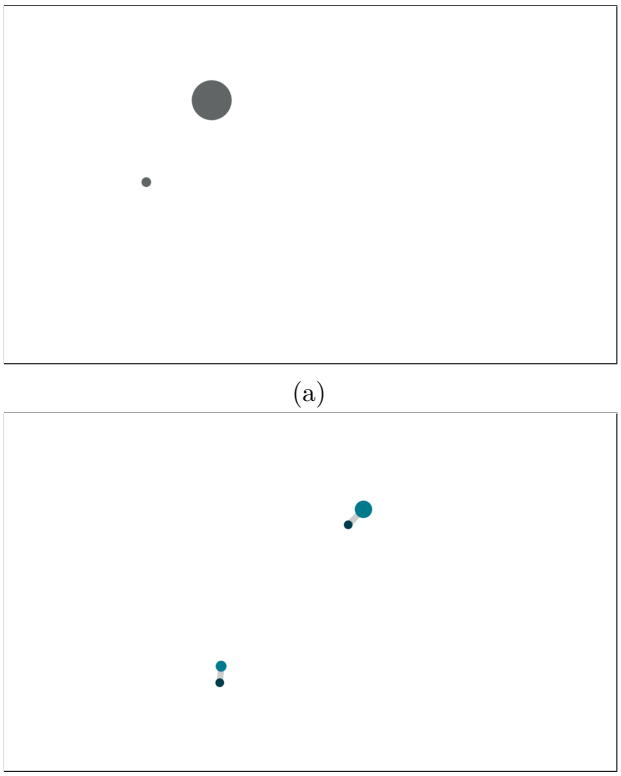
if it either contains a link affected by the highlighting or is incident to such a link. In Figure 5.9d the highlighting of the data links and their incident nodes can be seen.

5.2.4 Filtering

Filtering is a common interaction technique to explore a dataset. In out framework we implemented attribute-based filtering. This interaction removes all elements that do not fulfill the filter condition. Just like highlighting, filtering is propagated through the levels of the graph hierarchy. This means if at least one of the underlying data entities is not removed by the filter, the parent cluster or cluster link are rendered throughout the hierarchy.

In Figure 5.10, only relations, which are based on §4 of the Austrian *Medienkooperations- und -förderungs-Transparenzgesetz* [Med], are shown. Figure 5.10a shows the first level of the graph hierarchy, where a cluster is only rendered if any of the subnodes or sublinks fulfills the filter condition. Figure 5.10b shows the second level of the hierarchy, where the filtered entities can already be seen. The highlighting example in Figure 5.9c and 5.9d

was based on the same condition. We can see, that the highlighted entities correspond to the filtered entities in Figure 5.10.

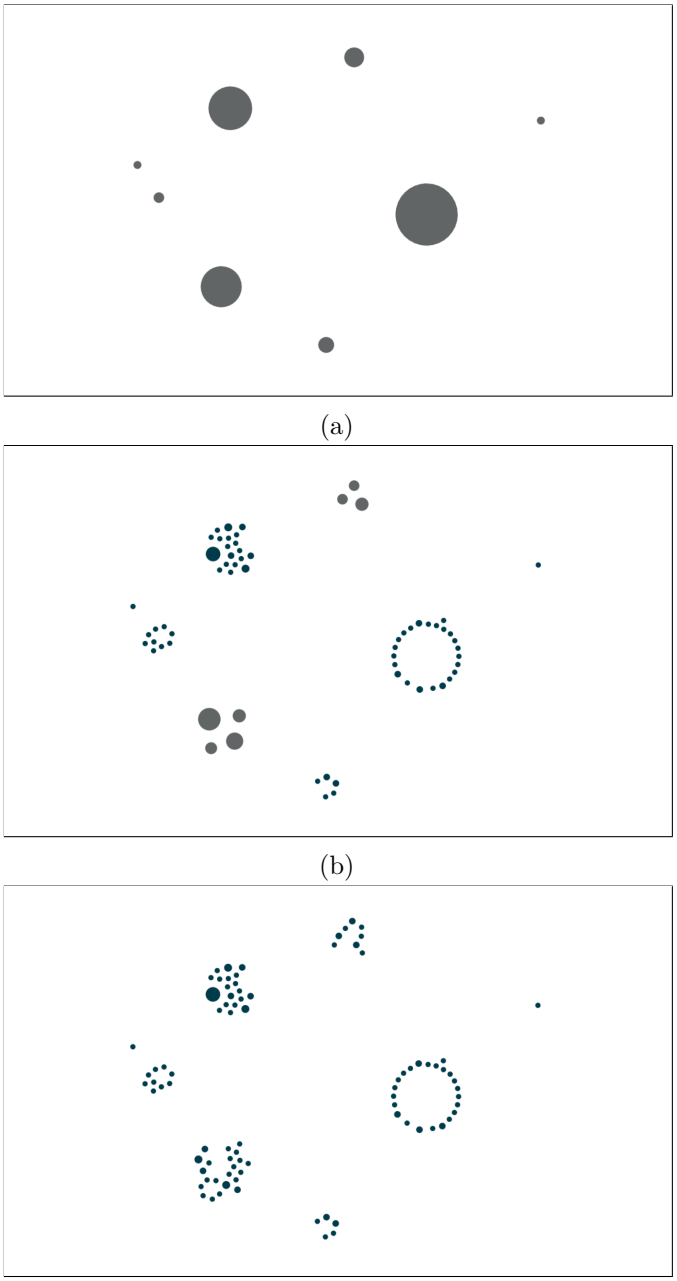


(a)

(b)

Figure 5.10: Example of attribute-based filtering of the test graph showing a subset of the Media Transparency Dataset through the levels of the hierarchy. Relations which are not based on §4 of the Austrian *Medienkooperations- und -förderungs-Transparenzgesetz* [Med] are filtered.

We also implemented a filter by the set of the nodes. The aim was to provide bipartite projections of the respective sets when selecting this set-based filter. However, due to the issues which arise by the number of links that need to be calculated for bipartite projections this concept did not make it into the version of the framework that is presented here, but it is considered as future work. Still, by selecting one of the two sets, all the nodes of the other set are hidden. Also, there are no links in the scene anymore, as by the definition of a bipartite graph there cannot be any edges between nodes of the same set. Although we do not show the links, this view can still help to investigate co-membership of set nodes inside clusters, based on the proximity in the layout. Figure 5.11 shows an example of the described, set-based, filter. In this example the test graph is filtered for the *Medium* set. The filtering is shown at all three levels of the hierarchy of the test graph.



(a) (b) (c)

Figure 5.11: Example of a filtering based on the set *Medium*, defined in the Media Transparency Database. (a) shows the first level, (b) the second and (c) the third level of the graph hierarchy.

5.3 User Interface

For some of the interactions that were described in the previous section, special user input is required. To filter and highlight the data the user needs to specify which attribute and which expression of this attribute should be used. In this subsection, we describe the user interface we implemented to carry out these interaction techniques.

Figure 5.12 shows the user interface. It consists of two main parts, the sidebar on the left side of the window and the HTML Canvas that covers the rest of the window. The visualization is rendered on the HTML Canvas and mouse events are registered to this DOM element.

The sidebar contains multiple collapsible sub-menus. The sub-menu called *Settings* contains debug settings. These are, for example, to disable seamless zoom and the animations between level switches. The zoom then just works without triggering level switches. Also rendering links can be disabled. Moreover, this sub-menu also contains progress bars for the layered layout calculation where the progress of each level simulation is shown. The *Data* sub-menu shows information of a node if one was selected on click.

The remaining two sub-menus are used to apply filtering and highlighting on the dataset. The process that is depicted in Figure 5.13, shows the interaction steps that are required to apply highlighting based on the node label *ORF eins*. To define a highlight, the user clicks on the *Highlight* header in the sidebar. There the user can click on the plus at the side of the respective attribute that he or she wants to choose. The example in Figure 5.13 shows the selection of the attribute *Name*. Now a text field opens up that allows entering the desired name. Moreover, suggestions based on the data are provided. After entering or selecting the label on click the highlighting is applied. The result of this highlighting through multiple levels is shown in Figure 5.9a and 5.9b. The process to apply a filter works in the same way as to apply a highlight, which is shown in Figure 5.13.

Figure 5.12: Framework with a sidebar on the left consisting of submenus for filtering, highlighting, and settings as well as one submenu which shows the data of an element if it was selected. The rest of the screen is covered with a canvas on which the visualization is rendered.

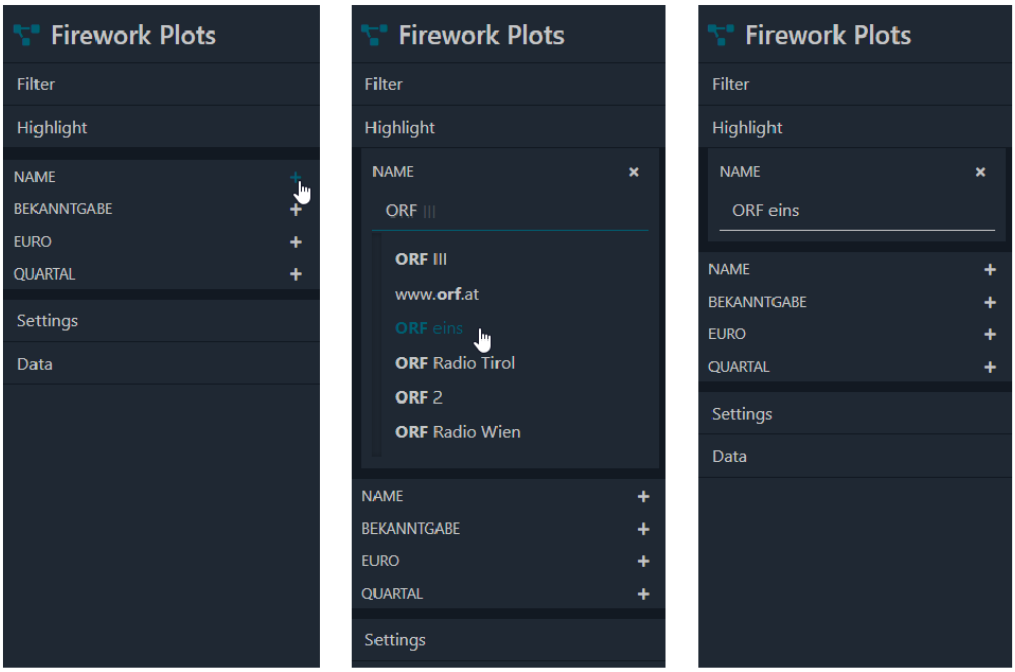


Figure 5.13: Steps to highlight the node by its label *ORF eins*.

CHAPTER 6

Implementation

For this thesis, we decided to implement a web application with a server-client structure. The main advantage of a server-client structure is that the server can be used such that computations on the client-side are reduced. The server is used for expensive calculations that only need to be done once. This is useful for data, which should be available fast, but can not be changed interactively. The client-side should only do calculations, which depend on user input and have to be done more often.

6.1 Server

The server is setup using Python and a Flask [Gri14] server. For the data structures and basic graph calculations, we use the NetworkX [HSS08] library. In the following, we are separating the tasks of the server into preprocessing of the data and serving the web application.

6.1.1 Preprocessing

We expect the data to be in CSV format, where each row represents a link. A few lines of the Media Transparency Dataset are shown in Table 6.1 and will be used to describe our assumptions about the structure of a dataset. One column defines the nodes of the first set of entities (*Rechtsträger*) and one column the set of the other entities (*Medium*). Moreover, we assume at least one more column defining the weight of the relations(*Euro*). If there are even more columns they are interpreted as attributes of the link, which can be used for filtering and highlighting in the client.

To set up the data as a NetworkX [HSS08] graph, we have to parse each row. In each row, we extract the set entities. If these nodes do not exist yet we add new nodes to the graph, which represent these entities and add a link between them. The link also gets

Rechtsträger	Quartal	Bekanntgabe	Leermeldung	Medium	Euro
Österreichischer Rundfunk	20184	2	0	www.youtube.at	5831.00
ÖBB-Werbung GmbH	20184	2	0	Österreich - oe24	12881.72
Ärztekammer für Wien	20184	2	0	gesund & fit	31000.00

Table 6.1: Example entries of the Media Transparency Database [Ope].

the respective weight assigned as well as a dictionary containing the additional attributes defined in the row.

For the agglomerative clustering we use the Python library *AgglomCluster* [Sea], which is an implementation of the approach by Newman et al. [New04], as described in Chapter 5. After clustering, we use the best number of clusters provided by the algorithm. If a cluster contains more than twelve nodes, we extract the subgraph that is defined by this cluster using the NetworkX function *Graph.subgraph*(), which takes the nodes as arguments. A subgraph is then clustered again until the clusters have a size smaller than twelve or the clustering algorithm cannot find any finer clustering.

From the clustering algorithm, we compute clusters of nodes at different levels. To create the path-preserving hierarchy we now have to calculate the links between those clusters for each level. To do so, for two clusters we collect all links where one endpoint is in one cluster and the other endpoint is in the other cluster. Based on all links between the nodes of two clusters a cluster link is created. Its weight is the sum of the weights of the links it represents. As the last step, we normalize the node and edge weights through the whole hierarchy.

6.1.2 Serving

Apart from the preprocessing step, the backend is further used to serve the web application. Graph layouts which were calculated on the client-side for the first time are sent to the server via a POST request and are stored, to avoid unnecessary recomputations and to reduce waiting times. The layouts are later retrieved from the client via a GET request. The data, which was already preprocessed to compute the path-preserving hierarchy, is also requested by the client using a GET request.

6.2 Client

On the client-side, we use JavaScript and realize 3D rendering with WebGL using the Three.js [Dan12] library. In Chapter 6 it was already shown that WebGL is the current state-of-the-art for the rendering of large graphs. As Three.js is a framework that allows an easy start but still offers low-level WebGL calls and to write shader programs, we decided to use this library. Although Three.js does provide an interaction system, it is not possible to add further functionality to a callback. For example, for the pan interaction, we want to hide the links. To call this behaviour, we still would have to implement another event listening system as it is not possible to add callbacks. Therefore, we decided

to use the event listener and the zoom behaviour of D3.js [BOH11] and handle the events on our own. Moreover, D3.js provides extensive and easy to use force-simulations. For the DOM manipulation and the POST and GET requests, we use jQuery [jQu]. The design and interaction elements are implemented using Bootstrap [Boo] in combination with Popper [Pop]. In the next paragraphs, we go into detail about the use of these libraries to implement our framework and performance measures. To do so, we split the next subsections into Rendering and Interaction. We discuss, which decisions lead us to the best performance we could achieve. We will describe our event handling, highlighting and filtering as well as decisions to improve the performance of them.

6.2.1 Rendering of the Firework Plots

In this subsection, we cover the implementation of the WebGL rendering part. We describe, which problem arose during implementation and how we tried to solve them. We separate this subsection into the aspects of rendering and creation of scene objects, the firework animations, calculation of the graph layout, and how we are handling the visibility management.

Since Three.js is a 3D rendering library, we embed our 2D graph into 3D space. The x- and y-coordinates are used for 2D positioning. The z-coordinate of the graph elements is always 0. To zoom in or out, the cameras z-coordinate is adapted. To position the camera, we first calculate the distance the camera needs to have to the graph such that all graph elements fit exactly into the screen space. We call this distance *graph distance* d_G and it is depicted in Figure 6.1 in a light gray. For the initial view, we position the camera at $1.5 \times d_G$ such that there is padding around the elements.

Scene Objects

The use of Three.js [Dan12] makes it easy to create the first scene as this framework provides predefined geometries, materials and meshes. Those predefined objects are focused on single elements like a circle, a sphere or a box, and using many of them is very expensive. To overcome this issue, Three.js implemented so-called buffer geometries, which are an abstraction of the vertex buffer objects that are uploaded to the GPU for rendering. Using a larger buffer minimizes the draw calls and, therefore, improves performance.

For our purpose, circles and lines are sufficient to represent entities and the relations between them. As circles are rendered using segments, rendering many of them leads to a lot of triangles and also decreases performance. We, therefore, use a buffer geometry where each position in the buffer defines the position of a node. We pass a texture showing a filled circle, which is then rendered at each node position. Three.js does provide a material which works as described and we will refer to this material as `Three Points`. The drawback of this material is that the size of nodes can only be specified as uniform, which means it is only defined once per buffer. As we want nodes with different sizes it is necessary to implement a custom material and shader program, which will be

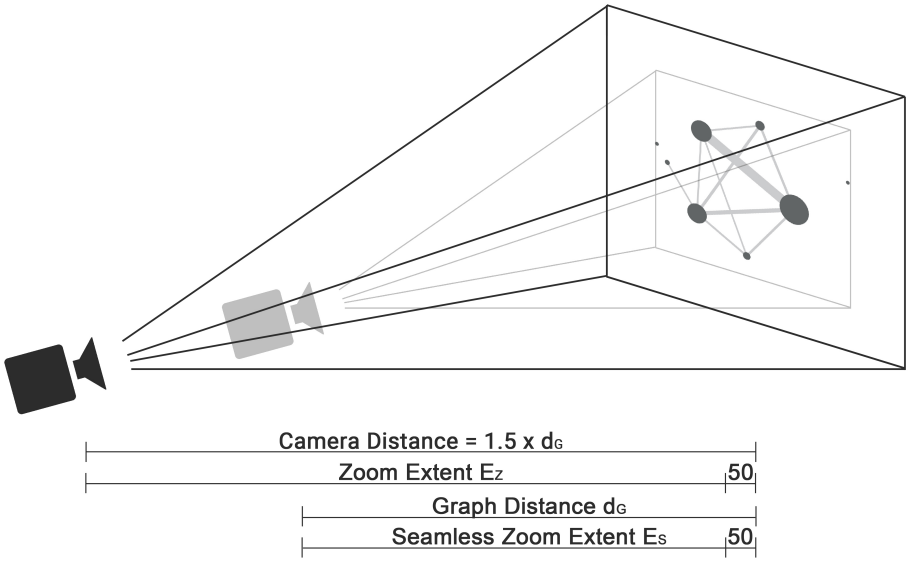
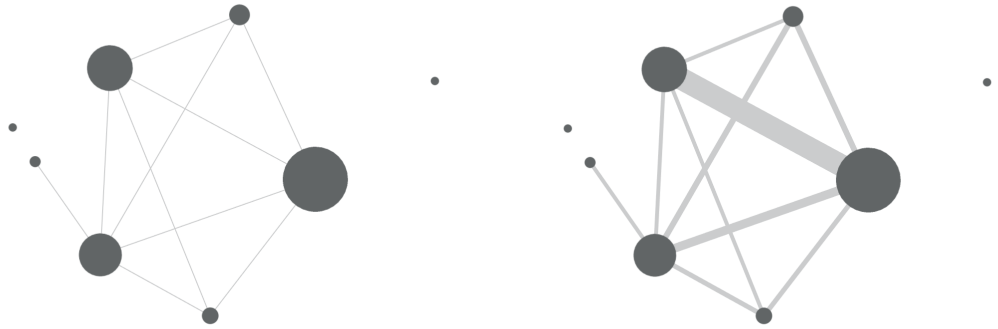


Figure 6.1: Setup of the 3D Scene. The camera distance d_G is defined as the distance of the camera to the graph elements such that all elements fit exactly into the screen. Also, the initial camera distance, as well as the zoom and seamless zoom extent, are shown.

called `Firework Node`. The sizes are now saved in an additional buffer and are passed to the shader as a vertex attribute.

As mentioned in Chapter 3, rendering lines with linewidths other than 1 is not straightforward. WebGL is based on OpenGL ES 2.0/ES 3.0 and its specification [LL19] states that the maximum and the minimum line width is only allowed to be 1.0. The visualization in Figure 6.2 shows the difference between rendering links with a linewidth of one and an actual linewidth bigger than that. One can see that this functionality can add important information to a visualization. To overcome this issue, the linewidths can be calculated in the shader programs that are executed on the GPU. The developer of Three.js published code implementing this approach in their examples [Lin], which will be referred to as `Three Line2`. In Chapter 3, we already found that the approach does not decrease performance a lot and, therefore, we decided to use this technique. As their approach also only allows linewidths to be specified per buffer and not per vertex, we extended the shader program, such that the specification of the linewidth per vertex is possible. We will refer to our custom line shader implementation as `Firework Link`.

To determine an appropriate buffer size we conducted a benchmark test where we rendered up to 5×10^6 nodes. The frame times of 100 frames were measured and the average is calculated across these measurements. The test was conducted on the high-end consumer machine described in Chapter 3.



(a) (b)

Figure 6.2: Test graph rendered with 1-pixel linewidth in (a) and the approach calculating linewidth in a shader program for comparison in (b).

A geometry holds the buffers, which store the positions of the nodes. If we use a small buffer size we need more geometries, if we use a large buffer size, fewer geometries are needed. For every geometry, a draw call to the GPU is executed. For the benchmark, we start with a buffer size of 10 and gradually increase it up to 5×10^6. The benchmark starts with many geometries and many draw calls and ends with only one geometry. From the size of the buffer s_b we can directly calculate the number of geometries in the scene with

$$\#Meshes = \frac{5 \times 10^6}{s_b}. \tag{6.1}$$

For $s_b = 10^3$, we have 5.000 geometries. `gl.drawArrays(mode, first, count)` is the WebGL function that is called to render vertices from a buffer array. The `count` variable is of type `long` specifying the number of indices to render. On a 32-bit architecture, 2^{32} and on a 64-bit architecture 2^{64} vertices can be accessed through indices and rendered. As such large buffers are possible by the API it is assumed that there actually is no drop in performance. In Figure 6.3 the results are shown. Since the performance is stable beginning with a buffer size bigger than 10^3 and does not decrease with a bigger size we can deduce that the performance does not depend on the buffer size itself but on the number of geometries. This is because, with more geometries, more expensive draw calls are made. Therefore, for the rendering performance, it is important to keep the number of meshes below 5000 on strong machines and even lower for average consumer machines.

Semantic Zoom

For the zooming events, we make use of the d3-zoom framework [d3-b], which allows us to call events on zoom start, zoom end and during zoom. To zoom into a graph, we

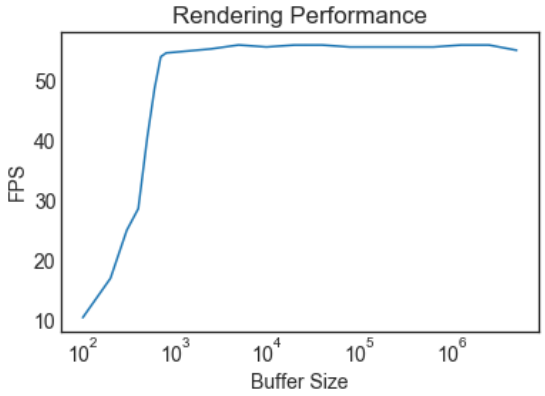
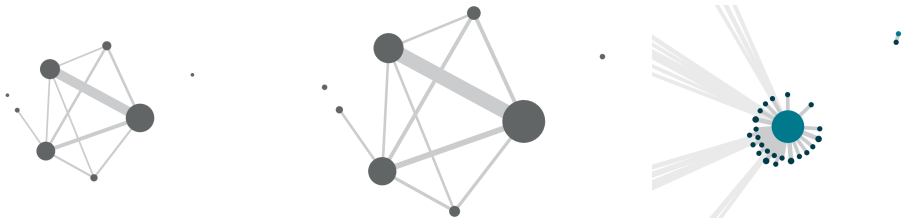


Figure 6.3: Plot showing the rendering performance with increasing buffer size. Note that a small buffer size means that many geometries are in the scene and a large buffer size means that there only is a small number of geometries in the scene.

change the z-position of the camera. For this, we define an extent for the z-axis between which the camera may move. We calculate the distance between the camera and the graph such that all objects fit into the view exactly. This is shown in Figure 6.1 as the graph distance d_G. The zoom extent E_z is then defined as $E_z = [50, d_G * 1.5]$. The camera distance of the initial view in the framework is set to $d_G * 1.5$. How the minimum and maximum zoom distances, as well as the graph distance d_G affect rendering the test graph can be seen in Figure 6.4



(a) (b) (c)

Figure 6.4: Images showing the zoom extent of our web application. (a) illustrates the maximum zoom distance, (b) uses the graph distance d_G and (c) is based on the minimum zoom distance.

For the seamless zoom itself, we define another extent, which specifies the camera distances between which seamless zoom is done. The first level switch is done at distance d_G, because at this distance the user already starts to focus onto a part of the visualization. Moreover, the last level switch is animated at the minimum zoom distance, since this shows the most details. We define the seamless zoom extent as $E_s = [50, d_G]$ as can be seen in Figure 6.1. To calculate the level switches on a logarithmic scale inside this

interval, we make use of the d3-scale [d3-a] library and apply the `d3.scaleLog()` function.

For the implementation of the firework animations, we have two components to describe. These are the opacity blending of nodes and links and the transitions between the positions.

The `Three Points` and `Three Line2` materials only lets you set the opacity per buffer as well. We added the functionality to do the blending per vertex. In Chapter 5 we described three cases of blending that can be applied to a vertex for the firework animations. We pass the type of blending as an integer to the shader. Also, a transition value between 0 and 1 is passed to the shader and is used together with the type of blending to calculate the respective opacity for each element.

Regarding the position transitions of the elements, the CPU-based approach would be to continuously manipulate the buffer attributes. This is expensive and the data on the GPU has to be permanently updated. We decided to use a GPU-based approach. As the layout is already precalculated, the different positions for each level will not change anymore. Therefore, in this approach, each geometry holds one position buffer for each level. For the level switch, only the position buffers of the involved two levels are needed. In the `Firework Node` and `Firework Link` programs, for each vertex a start position, which is the position in the previous level, and an end position, which is the position in the next level, are passed to the shader. With the transition value that was mentioned before we linearly interpolate between those positions during the animation.

Layered Force-Directed Layout

For the calculation of the force-directed layout, we use the extensive implementation in the D3.js [BOH11] library. NetworkX [HSS08] does provide layout algorithms, also including force-directed layouts, but the layout algorithms do not provide the possibility to define constraints. The implementation of the layered force-directed layout aims to allow the user to start the exploration of the dataset although the data is not even completely processed. We begin with the first level of the hierarchy, where we start a force simulation for this level. The positions of the scene elements after each iteration of the calculation are permanently updated and rendered to the HTML canvas until the simulation converges. After the first level is calculated, we start an HTML Web Worker whose task is to initialize and calculate the force simulation of the next level. After this web worker has finished its work and the resulting layout is sent to the main thread, the web worker for the next level is started until the layout for each level is calculated.

While the user starts the exploration and investigates clusters, their entities and sizes, and how they are connected in the first level, the layout calculation of lower levels are still running without blocking the user from his or her tasks. We only want the layout of a dataset to be calculated once. Therefore, after successfully calculating the layout, the data is sent to the server for each level. If a user requests this dataset at a later time,

there is no layout calculation needed anymore. This helps to even further reduce the waiting times for a user. A reason why the calculations are done on the client-side is that D3.js [BOH11] is a JavaScript library and implementing it in the backend is difficult. Moreover, layout recalculations based on user input are considered as future work. A client-side graph layout library, therefore, will be useful.

Regarding the constraints, we use the forces which are available in D3.js. For the **charge** force c_l for each level $l = 0..t$ we use the function `d3.forceManyBody()` and set its strength according to the formula:

$$c_l = -|E|a\frac{t-l}{t} \tag{6.2}$$

where a is a constant factor. A negative charge lets the nodes repel each other, whereas a positive charge lets the nodes attract each other. In the implementation, we found the best results with $a = 0.1$ however, getting an aesthetic layout may require to adapt this factor.

For the **collision avoidance** `d3.forceCollide()` is used. For each node, a radius representing its bounding sphere in the layout calculation needs to be defined. We specified the radius according to its calculated size.

The **attractive forces** are implemented using `d3.forceX()` and `d3.forceY()`. For each node both coordinates have to be set. If a node has a parent in the previous level, these are set to that position, else they default to the centre of the graph.

To specify **link distances**, `d3.forceLink()` is used. According to the description in Chapter 5 we use the inverse link weights to constrain the link length.

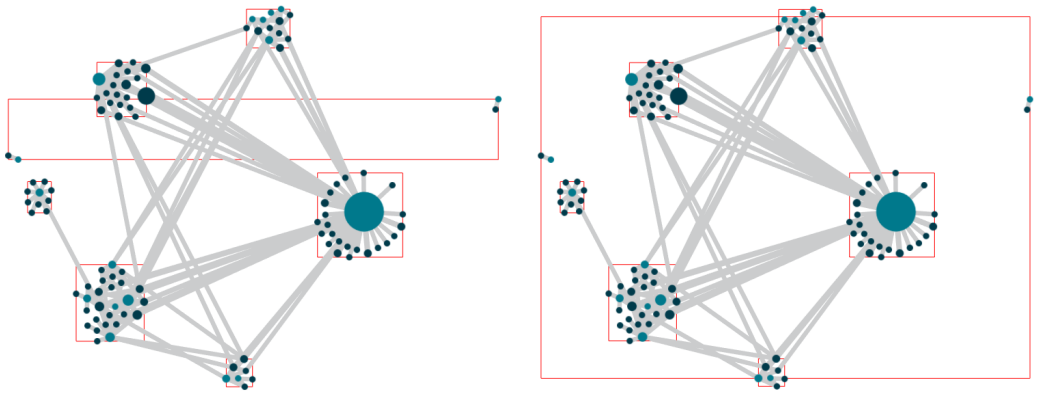
Visibility Management

In Chapter 5 we described the basic concept behind our visibility management. We discussed three cases, i.e., links whose endpoints are both visible, links where only one endpoint is visible and links where none of the two endpoints is visible.

As with the previous concepts, we also decided on a GPU based approach. To determine whether an endpoint is visible or not, we calculate its position in clip space and check if it is in the range $[-1, 1]$. If both endpoints are outside the clip space, we do not render the link. This effectively reduces clutter in our visualization. If only one of the endpoints is within the clip space, we assign a less prominent colour to the vertex and manipulate the z position of the vertex such that it is positioned and rendered behind links whose endpoints are both visible.

Moreover, we want to make use of frustum culling. We overcome the issue of constant node size or linewidth per buffer of the `Three Points` and `Three Line2` shader programs with the implementation of the `Firework Node` and `Firework Link` shaders. This allows us to define nodes and links of different size within the same buffer, ant to put

nodes and links into the same buffer that are spatially close. For each cluster of the first level, we have one buffer geometry for all the subnodes and one for all the sublinks containing their respective attributes. Due to the constraints of the layered force-directed layout we get meshes with small bounding boxes. Frustum culling geometries is more likely to be successful when zooming into the visualization. Clusters at the first level, which are isolated, are put into one buffer due to their small size and to avoid an unnecessary large number of meshes. We put all links, which connect nodes that are assigned to different clusters at the first level, into one big buffer. In Figure 6.5a we show the resulting bounding boxes of the node meshes and the bounding boxes of the link meshes are depicted in Figure 6.5b.



(a) (b)

Figure 6.5: (a) illustrates the bounding boxes of the node geometries and (b) the bounding boxes of the link geometries.

Context Pull

For the context pull interaction, we register an on-click event-listener from D3.js [BOH11] to the canvas. When a click event is triggered, the mouse position is passed to a callback function. The mouse position is transformed to normalized device coordinates, which are then passed to the raycaster implementation from Three.js [Dan12] to cast a ray through the scene at this position. The raycaster also works with points through which we defined our nodes. However, one has to set the precision of the raycaster for this type of element. We set the precision to 5 world units. If the raycaster does not return a node in the results, nothing happens. If the raycaster returns a node, it is used for the context pull.

For the context pull, we iterate over all links connected to the respective node and determine whether the other endpoint is inside the frustum or outside. This is done with the frustum implementation of Three.js. It provides the function `containsPoint()` that lets us check whether the endpoints of the links are inside. If an endpoint is outside the frustum, we calculate a new position inside the frustum. For this, we create a box

which is set depending on the size of the camera frustum. However, the left, right, top and bottom planes are padded by a constant of 10 world units. The link is then intersected with this padded frustum box. The intersection point is now inside the frustum with a distance to the edge of the view according to the padding. The node, as well as the respective endpoint of the link, are then positioned at this intersection point. This is done by manipulating the position buffer. To remove a selection, the user needs to click either on the same node again or select another node.

6.2.2 Interaction

The event system is implemented using D3.js [BOH11] event listeners. With the basic d3 event system, we bind mouse move, left mouse button down, left mouse button up, and left mouse button click events to the HTML Canvas. These events are used to trigger the pan, hover, and click, whose implementations are further described in the next paragraphs. As the semantic zoom, as well as the context pull, were described as part of the concept of firework animations, we already described their implementation in the previous subsection and will not further describe them here.

Pan

For our pan interaction, we bind the events mouse move, left mouse button down, and left mouse button up. If the left mouse button is clicked and stays clicked for the successively fired event, we start the pan interaction. The pan interaction itself does not influence the actual rendering performance as can be seen in Chapter 7. However, if the rendering of the mostly static scene does not perform at 60 FPS anymore, the lag becomes especially visible during user interaction. We decided to improve the performance during pan by not rendering the links. When starting the pan, the links are hidden from the scene. While the left mouse button is down, the user can drag the current view to the desired position. If the event for the left mouse button up is fired, links are rendered again. Three.js defines a `visible` flag for each geometry. This flag is checked before making the draw call to the GPU. If it is set to `false`, then the geometry is not rendered. This is the flag we set to hide or render the links.

Hover

To find elements in a 3D scene, raycasts are typically used, but they are expensive. As we are visualizing a 2D layout, a less expensive quadtree could be considered as well. However, this would mean that we have to use another data structure. Three.js does not provide a quadtree or octree implementation, but D3.js [BOH11] does. The D3.js implementation only lets one specify x and y coordinates. This is enough if we only want to hover nodes, but we also want to hover links. There also exist other JavaScript implementations for quadtrees. Many of those implementations let you specify bounding boxes by defining an x and y position as well as the width and the height. This means that the bounding boxes are axis-aligned. For the links in our visualization, which can

have every orientation in 2D space, this would mean that the bounding boxes are much larger than the actual link. We decided to stick to the Three.js raycasts as the results are precise, but to keep the number of raycasts as low as possible.

Whenever the mouse moves within the canvas, a mouse move event is fired. If the ray intersects the bounding box of a buffer geometry, it needs to be determined exactly which link or node is intersected. In this case, every item in the buffer is checked for a potential intersection with the ray. We decided to use large buffers since this is better for the rendering performance due to minimizing the draw calls, therefore, raycasting can get very expensive for large graphs. To reduce unnecessary raycastings through the scene, we only intersect the scene if the mouse position remains unchanged for a constant timespan. If the mouse move event is fired, we check whether the mouse position has changed. If it has changed, we use the DOM Window `setTimeout()` function, which takes a callback and the number of milliseconds as arguments. If the mouse position remains unchanged for the given milliseconds, then the callback function is executed and the scene is intersected with a ray. If the mouse position changes in the meantime, the timeout is cleared using `clearTimeout()` and, if necessary, a new timeout is created. Benchmarks showing the performance with and without this delay are shown and discussed in Chapter 7. If the user successfully hovers over a scene element, we highlight this element with the highlighting colour by manipulating the respective colour buffer. We show a label with some information about this element. To generate labels, we decided to create a small canvas, draw text to it, and use it as a texture with the Three.js sprite implementation. Although text geometry does exist in Three.js, it is very expensive, as a text geometry contains a lot of vertices and is not the best choice, if performance is important. The labels were implemented as described, because at the beginning of the thesis it was not clear, whether we would show the labels on-demand or not. Using a HTML tag may seem the better way if only one element is labelled, however, if many labels were added to the scene this also means adding many DOM elements, which is poor in performance. The current implementation allows better performance for future work regarding labelling.

Highlighting and Filtering

To implement highlighting and filtering, we check the attributes for their properties. First, we distinguish whether an attribute can be interpreted as a number or a string. Apart from that, we check how many characteristic values appear for this attribute. If it is below a threshold, we allow the user to select the attribute out of a dropdown menu with all possible values, independent of the data type. For a larger number of characteristic values, we separate between numbers and strings. For numbers, we add a range slider to the user interface where the user can select a minimum and a maximum value to highlight or filter the data. For strings, we add a text field to the interface, where the user can enter the value to filter or highlight for. To support the user with this task, we use the JavaScript library typeahead [Fau], which offers suggestions based on at least three letters typed into the text field.

When the user successfully selected a filter or a highlight, we iterate over the data nodes and links to select the respective data. Those results are then propagated to the parent cluster nodes and links. This means, if all subnodes of a cluster are filtered out by the selected attribute then this cluster is also not shown. If there is at least one item still visible, then also the parent elements stay visible.

CHAPTER 7

Results and Evaluation

The purpose of this chapter is to demonstrate, which problems we can solve with our framework. With the help of use cases, we will show how our framework can handle perceptual scalability issues. We discuss, which common graph analysis tasks can be fulfilled. In a comparison with related work, we discuss which measures we implemented to overcome the drawbacks of the related work, especially regarding the understandability of the visualization. Additionally, we show the results of some benchmarks regarding the performance of our framework and our measures that handle the interactive scalability challenge.

7.1 Use Cases

At first, two use cases with different datasets are described to give better insights into our framework and its capabilities. The datasets vary in size, as well as in their graph properties. Both graphs are, by the definition of Yoghourdjian et al. [YAD+19] (see Chapter 2) very large and dense. We want to show how the orientation of the user is supported and how the user can carry out common graph analysis tasks. As described in the introduction in Chapter 1, our focus is on topology-based tasks where the user tries to get information on the structural characteristics of the dataset. We also want to consider other graph analysis tasks as described by Lee et al. [LPP+06]:

- **Topology-Based Tasks**
  Find adjacent nodes and, more generally, accessible nodes of a node.
  Find clusters and connected components.

- **Attribute-Based Tasks**
  Find nodes and links based on attributes.

- **Browsing Tasks**
  Follow a path and revisit nodes of such a path.
- **Overview Tasks**
  Estimate the size of a network.
  Find patterns in an overview, i.e., clusters and connected components.
- **High-Level Tasks**
  Compare two networks.
  Determine how a graph changed over time.

7.1.1 Media Transparency Database

The media transparency database defines the two sets of entities as *legal entities* and *media organizations*. It consists of 1393 legal entities and 4532 media organizations. Between the entities of these two sets exist 17780 links. A more detailed description of the Media Transparency Database can be found in Chapter 2.

In the following figures, grey circles represent clusters, circles in light blue represent legal entities, and those in dark blue represent media organizations. In the preprocessing step a graph hierarchy with six levels was generated. Table 7.1 shows how many cluster nodes, cluster links, data nodes and data links exist on each level. Links which connect a cluster node with a data node are counted as cluster links.

Level	Cluster Nodes	Cluster Links	Data Nodes	Data Links
1	85	141	0	0
2	111	1089	242	174
3	212	4424	2144	2141
4	246	7856	4016	5330
5	79	3811	5523	13145
6	0	0	5925	17780

Table 7.1: The number of graph elements at each level of the Media Transparency Database.

Figure 7.1 shows the first level of the graph hierarchy. This is also the first part of the visualization a user sees. It already gives an overview of the basic structure of the graph and components of the dataset. This overview shows the coarsest clusters of the dataset and also how they are connected.

The small circles at the right of the image are isolated clusters. Due to the path-preserving hierarchy, it can be assumed that the entities which are represented by these clusters are only connected with entities in the same cluster. By the size of the clusters, it can already be said that they only represent small amounts of the advertisement spendings. By investigating them one-by-one it can be seen that the clusters only contain between two to four nodes and their relations. One cluster, for example, is formed by the legal entity *Landesgremium Wien des Parfümerie- und Drogerie-Einzelhandels* and the media

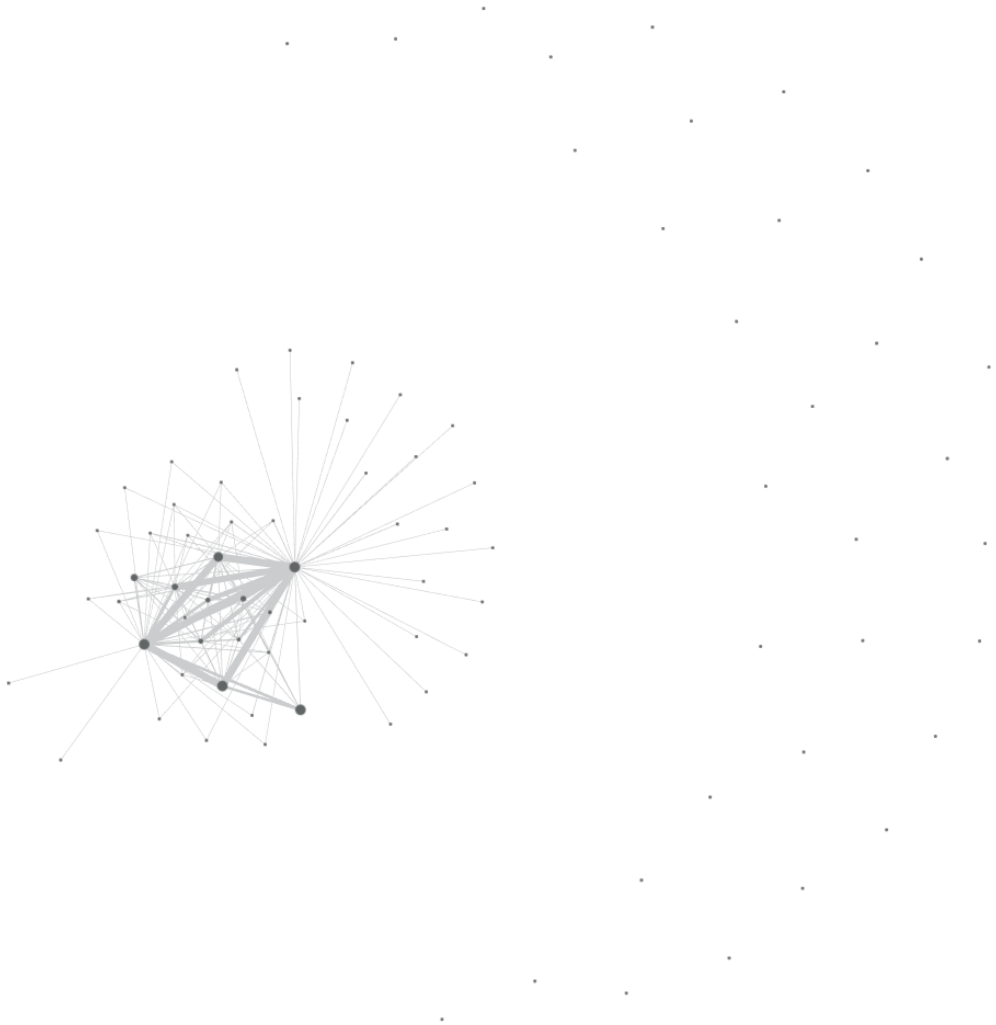


Figure 7.1: The initial overview of the visualization. It shows the first, coarsest level of the graph hierarchy. Small isolated clusters can be seen at the right and at the top. The main component is depicted in the centre.

organization *www.meineparfumerie.at*. Another cluster consists of the *Ars Electronica*, a centre for electronic arts in Linz and the *Design Center Linz*. In this cluster, *Ars Electronica* is a media organization receiving money from *Design Center Linz*. *Ars Electronica* also functions as a legal entity, but the node representing it as an entity of this set is located within another cluster as we are assuming the sets to be disjunct.

The main component with the star layout seems to be of much more interest. It represents the largest part of the dataset and according to the path-preserving hierarchy, all of

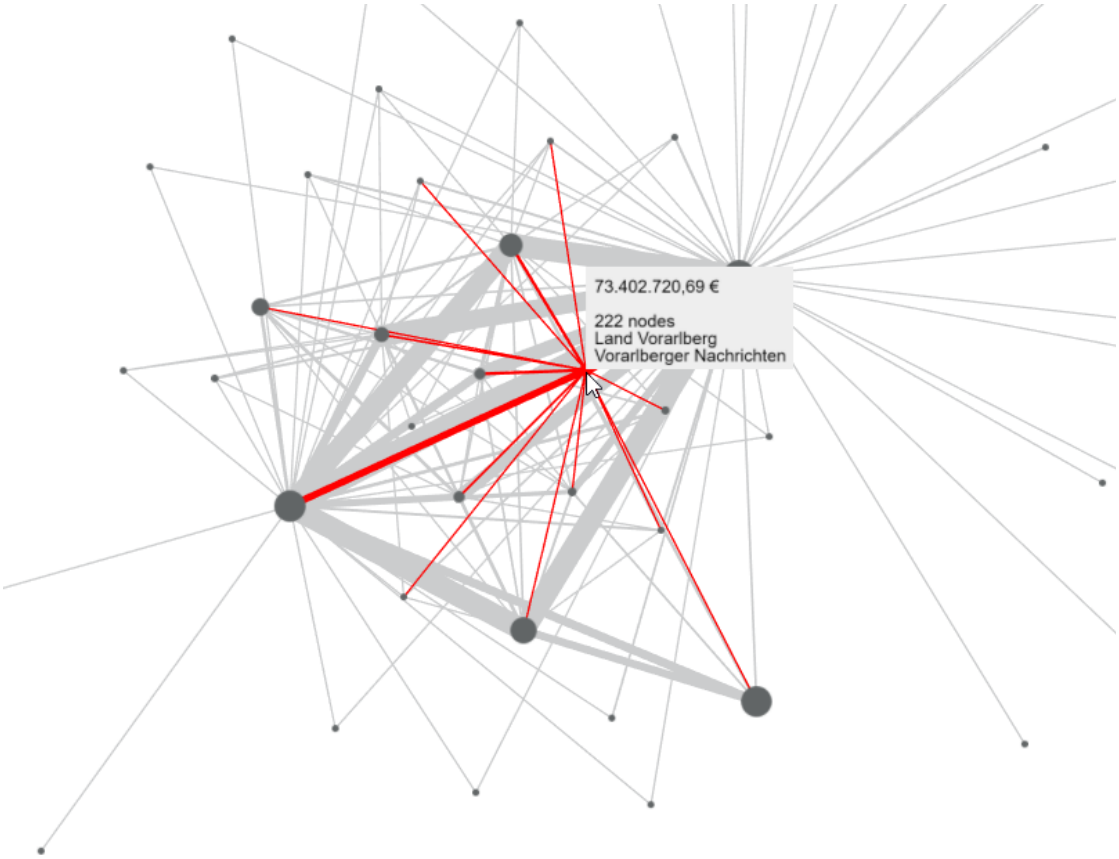


Figure 7.2: When hovering a node, a label is shown and the hovered node and all connected links are highlighted. The label shows the total sum of the cluster from the underlying data nodes, as well as the number of data nodes it represents. The largest entities from each of the two sets in the last two lines of the label.

the entities within these clusters are connected. This helps to answer accessibility tasks because the user knows that every node from within such a structure can be reached.

From the layout, one can see that there are many clusters further away from the centre of this component. This means that these clusters only have a few small connections to entities of the clusters in the centre. As visible from their size, they represent only a few entites. In general, clusters closer to the centre have more and stronger relations with each other than those with larger distance to the centre. This observation is emphasized by the linewidth that is larger for the edges in the centre. Also, the clusters are larger.

If a node is hovered, all incident links are highlighted and a label is shown. The highlighting helps the user to fulfil topology-based tasks [LPP+06] like finding adjacent nodes. The label states the sum of the relations represented behind these clusters as well as the number of data nodes it comprises. It also gives the largest entity of each of

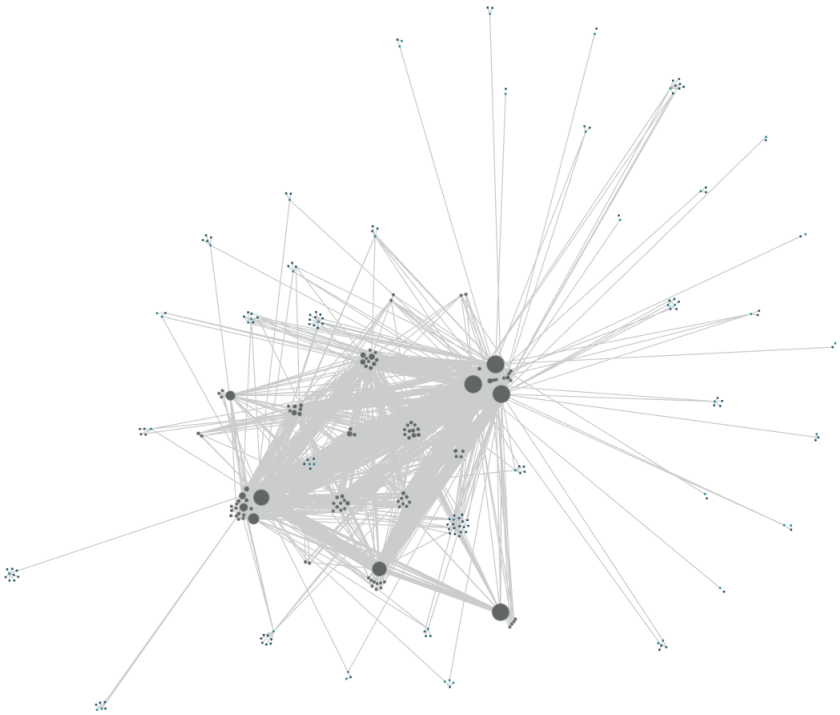


Figure 7.3: The visualization right after zooming in and rendering the firework animations. The clusters in the centre reveal multiple sub-clusters. The isolated clusters at the edge of the image as well as the smaller clusters in the main component already reveal data nodes.

the two sets. This does not necessarily mean that these largest entities are connected as well. In Figure 7.2, a cluster node at the centre is hovered and the label shows that this cluster represents a total amount of spendings of more than 73 million Euros. It consists of 222 data nodes. The largest legal entity within this cluster is *Land Vorarlberg* and the largest media organization is *Vorarlberger Nachrichten*. In some cases, we can infer semantic relations from the computed clusters. In this case, the geographical proximity is obvious. One can expect that there are going to be more entities in this cluster, which are related to Vorarlberg. This is discussed with the investigation of the next levels in the following paragraphs. Another example is a cluster where the largest entities are *Flughafen Wien Aktiengesellschaft* and *Airline Business*, which leads to the assumption that this cluster might contain more entities related to air travelling.

To start the investigation, one makes use of the seamless zoom and starts zooming into the graph. Figure 7.3 shows the second level of the graph hierarchy using seamless zoom. It can be seen that the larger clusters at the centre comprise multiple smaller clusters. The smaller clusters, which are further away from the centre, already depict leaf nodes,

i.e., the actual non-aggregated data nodes. The smaller clusters that are opened to reveal the data nodes at this level show similar patterns as the isolated clusters from the top level. They comprise a few small entities and also reveal some meaningful relations. For example, one cluster contains the entities *business guide*, *invent* and *Österreichisches Patentamt*.

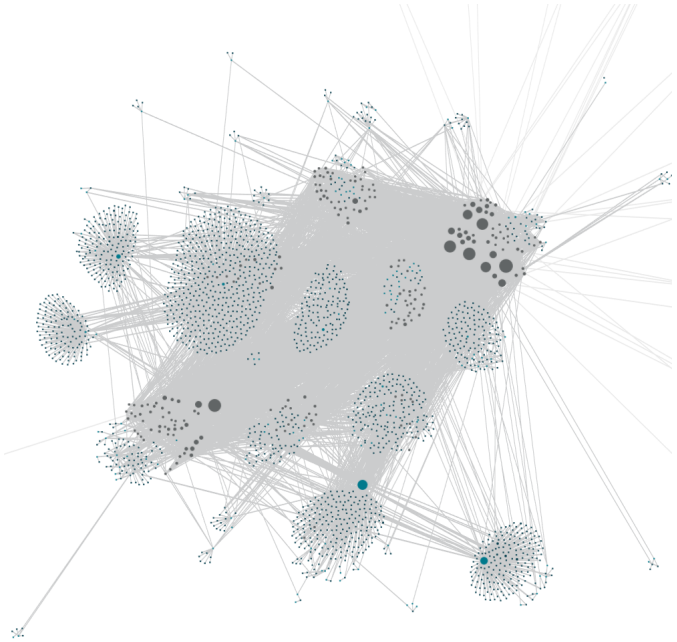
Looking at the sub nodes at the second level of the Vorarlberg cluster at the first level, one can see multiple smaller clusters. From their labels, the geographical connection to Vorarlberg is still obvious. For example, there is one cluster where the legal entity *Land Vorarlberg* and the media organization *Feldkircher Anzeiger* are the largest set entities. *Vorarlberger Kraftwerke Aktiengesellschaft* as legal entity and *ORF Radio Vorarlberg* as media organization are the listed entities on the label. Many of the other clusters show a similar relation to Vorarlberg but there are also a few clusters that do not seem to have an obvious connection to Vorarlberg. For example, the cluster with the largest entities *Universität Linz* and *www.absolventen.at*, does not have an obvious relation to Vorarlberg.

Zooming into the next level, which is given in Figure 7.4, many data nodes are shown which were still aggregated at the previous level. Many fan structures are visible as many of the clusters consist of one larger legal entity spending money on many different small media organizations. In Figure 7.4b, the largest visible media organization at this level is hovered. This entity is *Stadt Wien*, which paid more than 185 millions of Euros to different media organizations in the years from 2012 to 2019. The parent cluster of *Stadt Wien* was formed only by *Stadt Wien* and many small media organizations, which only received advertisement money from this public authority in the visualized time period. There are many relations to other entities and clusters as well.

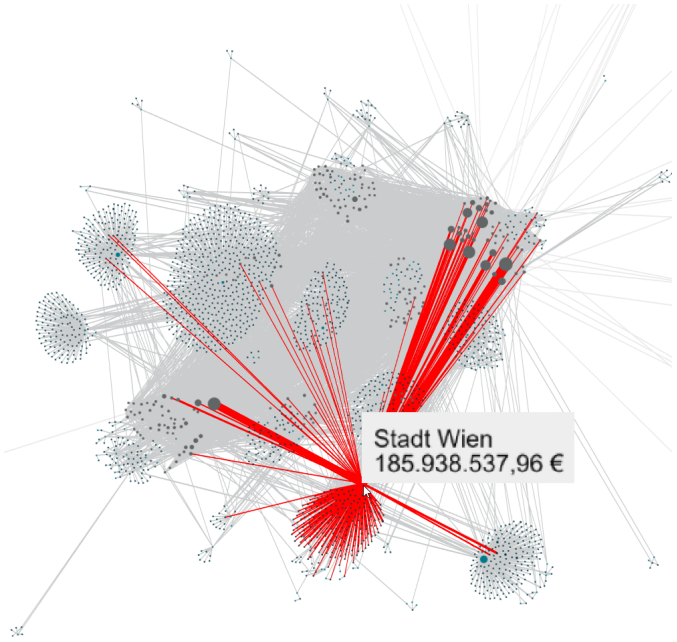
When continuing the exploration, one can see that a few data nodes are already revealed but that there are still a lot of cluster nodes. These clusters correspond to further nodes and edges with lower advertisement sums. For example, there is a cluster with the entities *Stadtgemeinde Bregenz* and *Bregenzer Blättle*. Many clusters containing different economic chambers of Vorarlberg can be found.

In the next level in Figure 7.5, the first larger media organization is revealed as a dark blue circle at the bottom left corner. This node represents the media organization *ORF2*. Apart from this, the highlighting of *Kleine Zeitung* is visible, which is also a larger media organization. It received more than 43 million Euros from different legal entities. According to the firework animations, most of the related nodes are in the same hierarchy branch, or other clusters with many entities related to *Kleine Zeitung* are located in the vicinity. There are also a few connections to other entities and clusters. At this level, the data nodes of the Vorarlberg cluster are fully revealed. The largest node shows the media organization *Vorarlberger Nachrichten*. Most of the nodes in the Vorarlberg cluster are connected to this entity.

Figure 7.6 depicts the fifth level of the hierarchy. The biggest entity revealed from the previous level is *Kronen Zeitung*, at the left of the image. 147 million Euros were paid



(a)



(b)

Figure 7.4: The third level of the graph hierarchy, which can be seen in (a), already depicts larger groups of data entities according to their clustering. (b) shows the on-hover highlighting of the largest legal entity, which is visible at the third level.

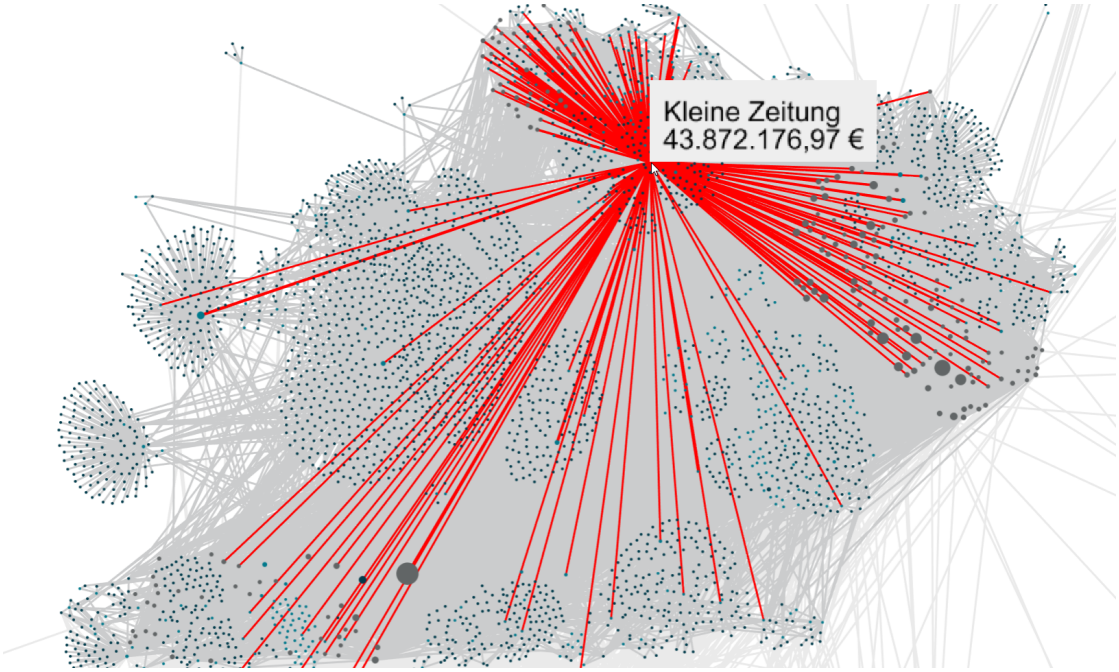


Figure 7.5: Level four of the graph hierarchy where larger media organizations start to be shown. One of them is hovered to communicate how it is connected with entities of other clusters.

for advertisements to this media organization over the past years. By investigating it, one can see that it has many connections to all the other clusters. The largest relation is the one to *Stadt Wien* (at the bottom of the image), which represents more than 25 million Euros. There are also other well-known media organizations visible, which are not as large as *Kronen Zeitung*. On the right side, we can see dark blue nodes representing organizations like *Heute*, *Österreich*, or *Kurier*. Also, larger legal entities like *Bundesministerium für Finanzen* or *Wirtschaftskammer Wien* are shown at this level. Most of the data nodes are already revealed. There are only a few clusters left at the top right corner, which will be opened in the next level. According to the clustering algorithm, the nodes within these clusters have more relations with each other than to other neighbours. Therefore, these nodes are stronger connected. Apart from the data insights, at this level, the influence of visibility management becomes apparent, as edges leaving the context are rendered in a less prominent colour to reduce clutter.

Figure 7.7 shows parts of the last level using seamless zooming. The larger dark blue nodes represent different well-known media organizations like *Die Presse* or *Der Standard*. As there are multiple larger media organizations in this area, this is a very dense part of the graph. When further investigating, it can be seen that the just revealed data nodes *Die Presse* and *Der Standard* are related to many universities and other academic organizations, and that they are within the same clusters. These are, for

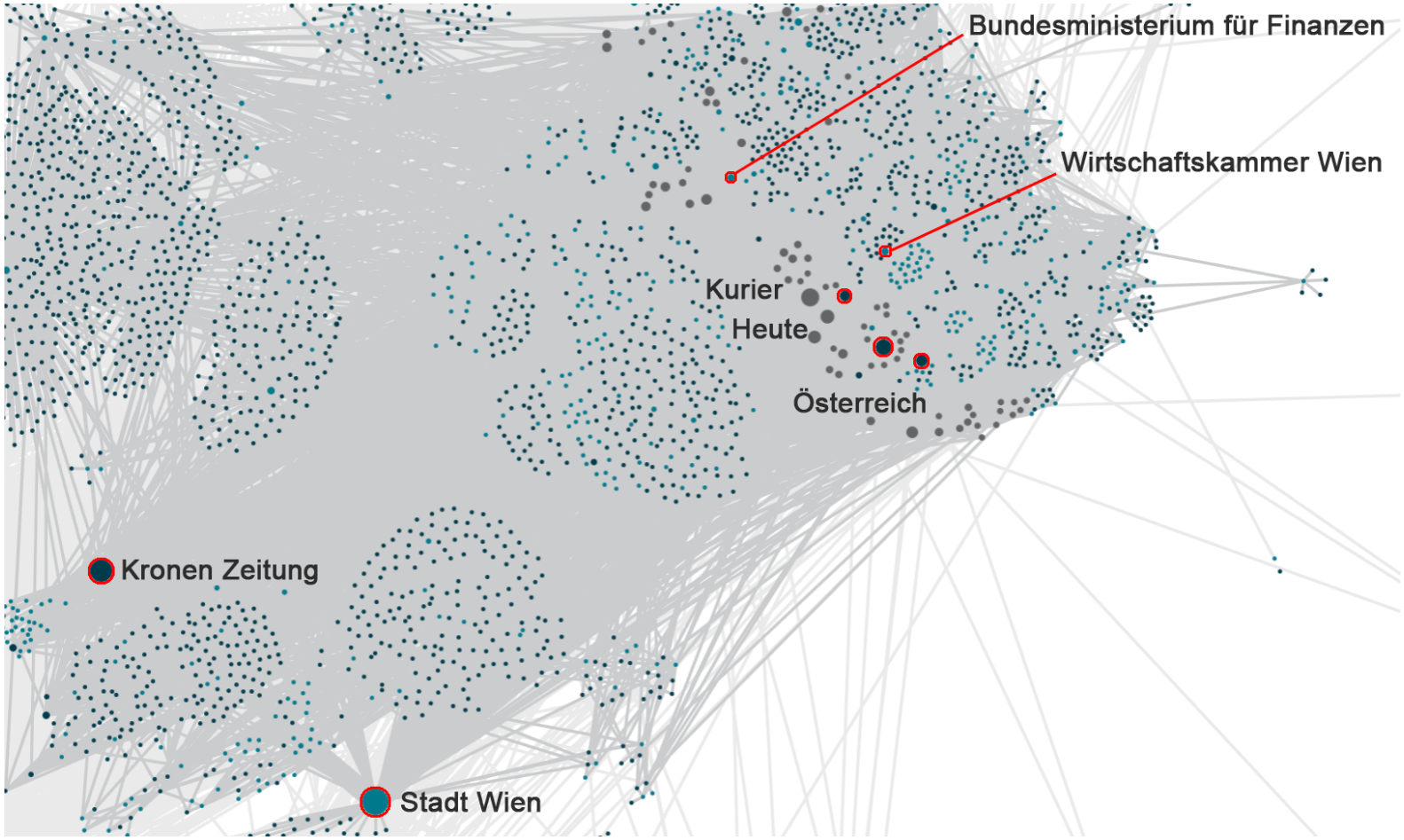


Figure 7.6: Part of the visualization which is obtained through seamless zoom at the fifth level with annotations. Almost all entities are already shown apart from a few clusters. The visibility management can already be seen as edges leaving the view are rendered in a less prominent colour.

Figure 7.7: Visualization of a zoomed view of the last level with annotations. All clusters are finally revealed. The clustering is still visible due to the positioning of child nodes based on their parent cluster's position.

example, *Alumniverband der Universität Wien*, *TU Career Center*, *Veterinärmedizinische Universität Wien* and *Medizinische Universität Wien*.

In the previous paragraphs, the use case was described based on the exploration of the topological features provided by the visualization itself, the hierarchy structure, and the layered layout. The framework also provides sophisticated functionality for attribute-based tasks [LPP+06]. For example, the user can apply highlighting to elements of the graph based on attributes which are provided in the dataset. To present an example for this use case, we assume a user wants to find the entity *Wiener Tourismusverband*. The user can select a highlight based on the defined node labels. The highlighting of the entity is propagated through the hierarchy such that at each level the data node or the cluster node containing the data node is highlighted as well. Figure 7.8 shows the highlighting in the first and the last level.

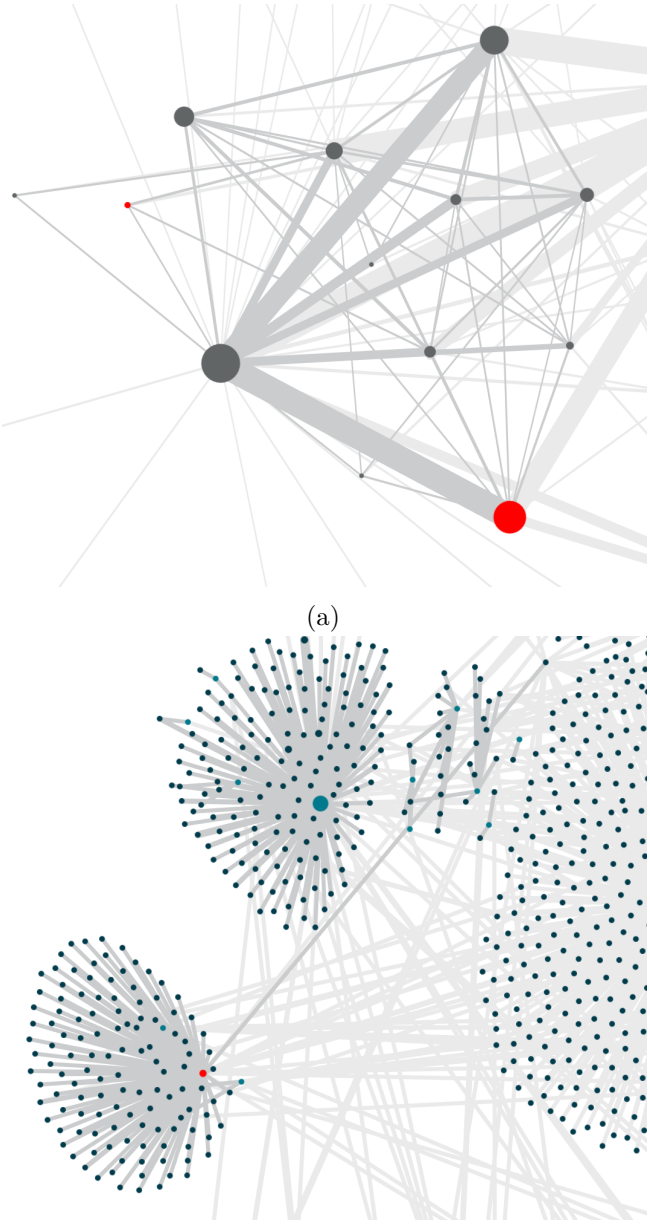
Starting from the first level, the user knows where he or she has to zoom into to reveal the desired entity, as the parent cluster is highlighted. This way, the user also investigates the structure down to the nodes. In the provided case in Figure 7.8, two nodes are highlighted. This is because in this dataset entities can be both media organizations and legal entities, but as we assume disjunct sets in our bipartite graph they are handled as two different entries. For the provided example, we zoomed into the smaller, highlighted, cluster at the top left. Highlighting works for all attributes defined in the dataset. In the context of this use case, the highlighting can, for example, also be applied based on the *Bekanntgabe* attribute of the dataset. This attribute provides the information on which paragraph of the law the respective publication of advertisement spendings is based.

After the highlighted entity has been found and zoomed, one also might want to see the neighbours of this node. To do so, the user can click on a node to select it and apply context pull. All the neighbours, which are connected to the selected node, but are not within the current context, are pulled into the context. In Figure 7.9a, the image illustrates a highlighting on hovering of the entity before the context pull is applied. Figure 7.9b shows the visualization after context pull is applied to the node of *Wiener Tourismusverband*. It can be seen that the neighbours are now visible, and the links are cut. Hovering the elements of the current selection, the relations to direct neighbours can be further investigated. The context pull is removed by either clicking the node again or selecting another node for context pull.

The framework also provides filtering of the data as attribute-based tasks for graph analysis. In Figure 7.10 the filtering of relations whose weights are larger than 5 million Euros is shown. Figure 7.10b shows the result at the first level and Figure 7.10d the result at the last level. To compare the results, also the unfiltered data is shown at the first level in Figure 7.10a and at the last level in Figure 7.10c.

7.1.2 Medical Dataset

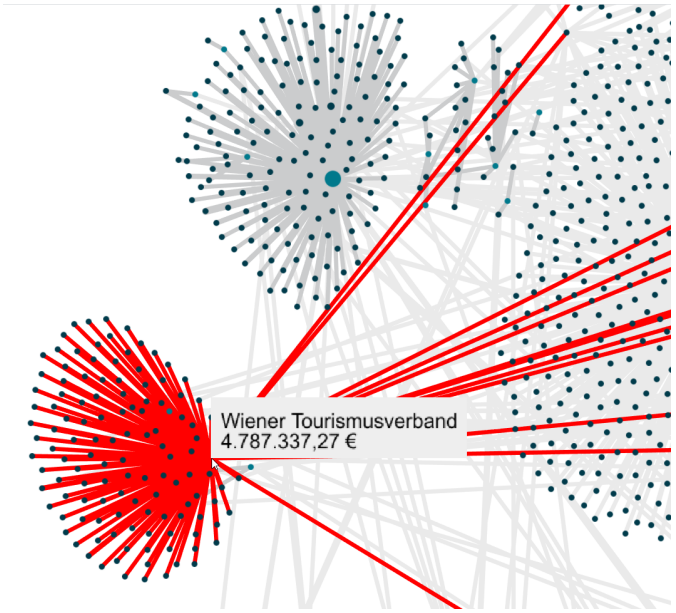
The medical dataset is larger and denser than the previous use case. Since we do not have permission to show any insights into this dataset, we will only shortly discuss this example



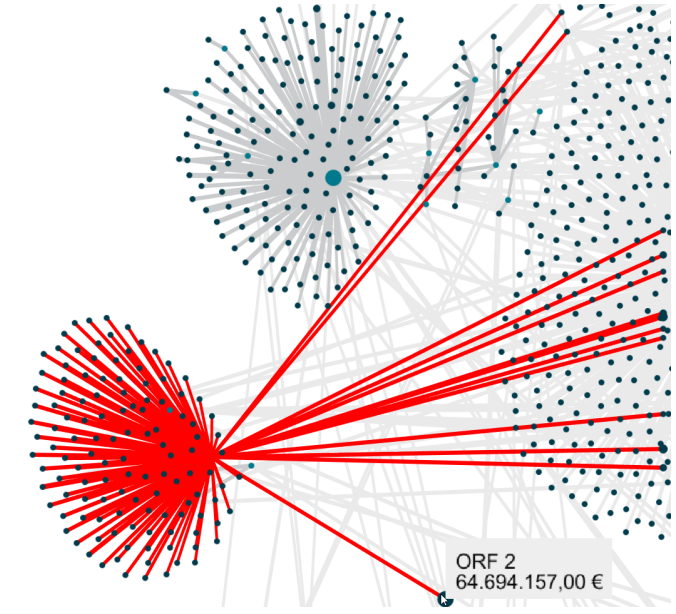
(a)

(b)

Figure 7.8: Two images depicting the highlight propagation of a selected entity. (a) shows the highlighting at the first level and (b) the highlighting at the last level. The entity is highlighted by its label. The highlighted entity appears once as legal entity and once as media organization. Therefore, in (a) a smaller, highlighted, cluster at the top left and a larger, highlighted, cluster at the bottom right, are visible. In (b), we can see the zoomed in view into the smaller cluster at the top.



(a)



(b)

Figure 7.9: Visualization showing an example of context pull. In (a), one can see the highlighting of an entity and its relation by hovering over it. The links also show relations to entities which are not in the context. In (b), the context pull of the respective entity is shown where the neighbours are pulled inside the context.

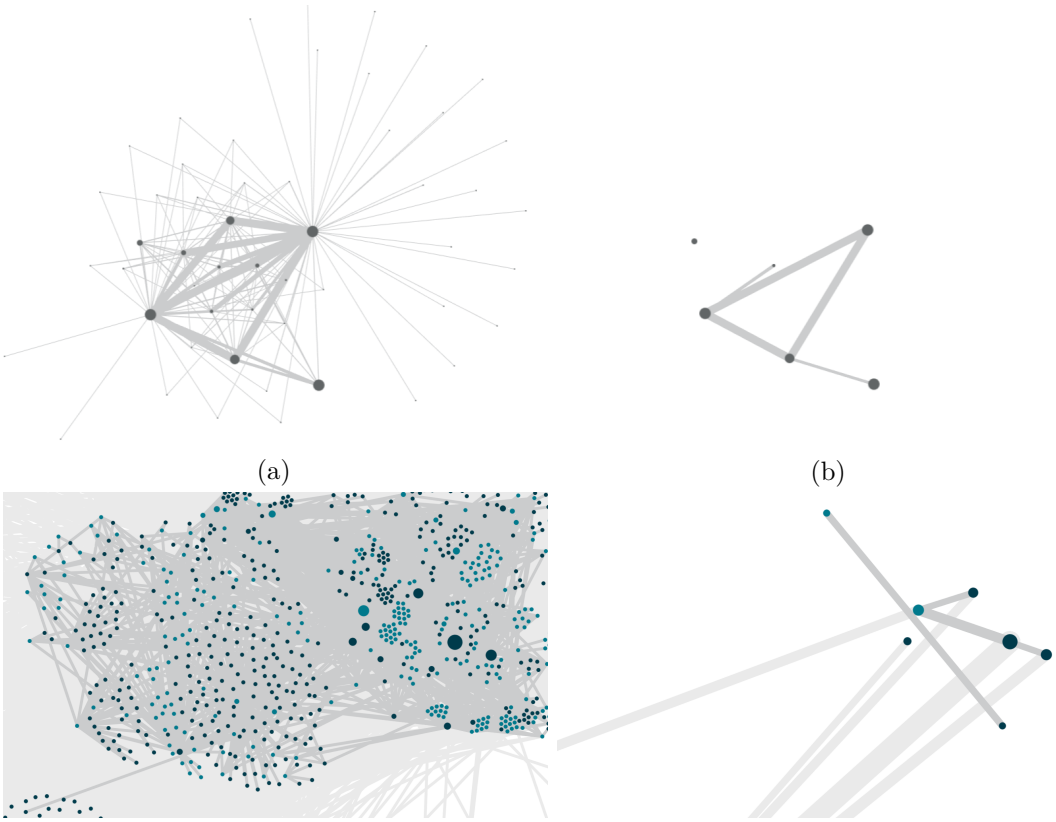


Figure 7.10: Filtering of the Media Transparency Database by the edge weights showing only edges with a weight bigger than 5 million Euros. (b) shows the filtering results of the first level, (d) depcits the results at the last level. (a) and (c) show the same views at the same levels but without the filtering.

based on its high-level features. Labels which are shown on hover are manipulated such that no detailed information is given. The dataset consists of 1147 entities in the first set and 3269 entities in the second set between which 88367 links exist. A graph hierarchy with nine levels was generated. Table 7.2 holds the number of elements per level.

The first level of the hierarchy, which is also the initial view when starting the framework, is shown in Figure 7.11. In the overview, it is visible that there are three connected components. Two of them are very small, only consisting of two subnodes each. The third component represents the rest of the dataset. In this component, there are three big clusters, which are incident to the largest links. This indicates that these clusters and relations represent the largest portion of the dataset. Also, there are multiple smaller clusters, which are connected to these larger clusters.

In Figure 7.12 levels two to five are illustrated when using seamless zoom. The second

Level	Cluster Nodes	Cluster Links	Data Nodes	Data Links
1	19	30	0	0
2	31	217	41	25
3	58	1062	99	62
4	129	4978	660	681
5	155	16822	2076	7052
6	136	26115	3192	21816
7	35	11057	4228	67721
8	3	1607	4401	85570
9	0	0	4416	88367

Table 7.2: The number of graph elements at each level of the medical dataset.

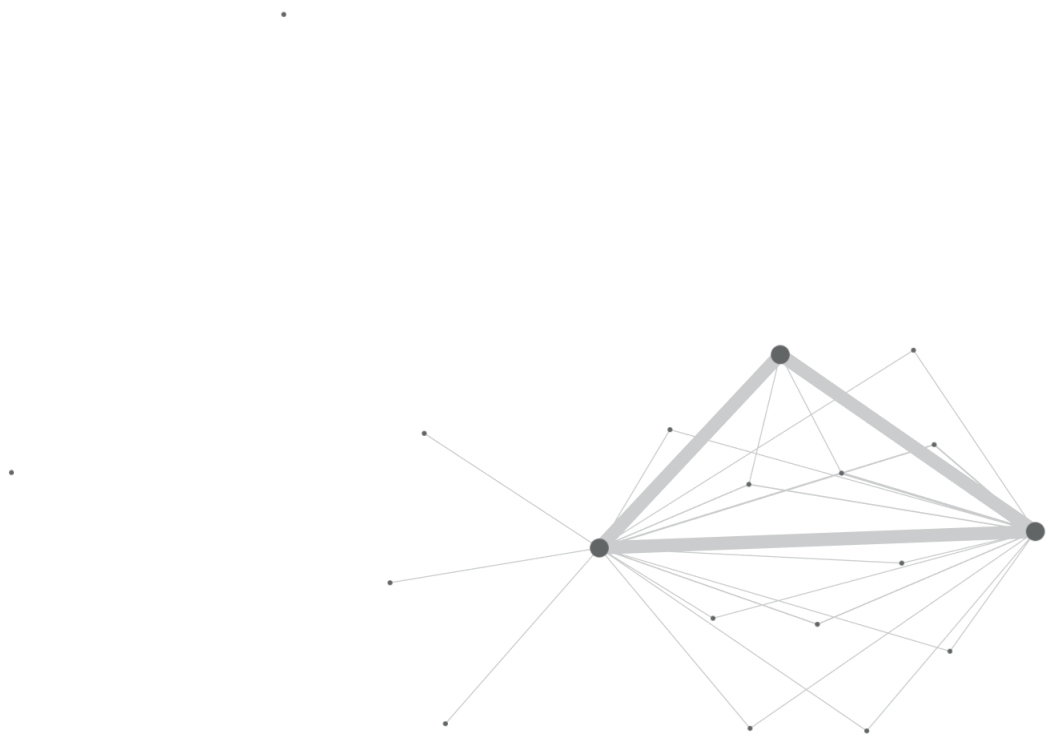


Figure 7.11: The first level of the graph hierarchy and initial visualization in our framework of the medical dataset.

level, in Figure 7.12a, shows that the small clusters of the big component are already open, whereas the larger clusters are divided into multiple smaller clusters. The same holds for the third level in Figure 7.12b.

In the fourth level (Figure 7.12c), the entities of the first big cluster, are revealed. This cluster can be seen at the left of the image, where one large entity of the first set in light blue is visible surrounded by many small entities of the second set in dark blue. Through

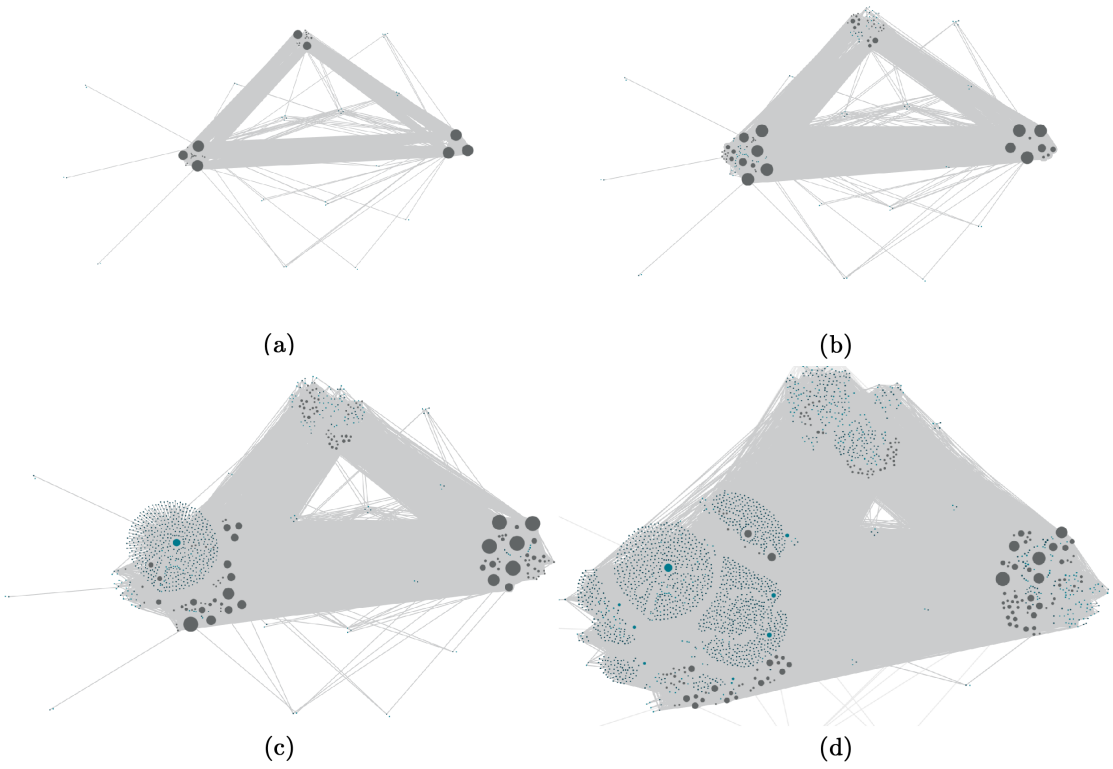


Figure 7.12: Views at levels two to five of the medical dataset using seamless zoom.

the hierarchy, multiple of such structures open up. Levels six to nine are illustrated in Figure 7.13.

In the last two levels, which are depicted in Figure 7.13c and Figure 7.13d, we can see that even in the zoomed view the links are still not distinguishable. This can be seen, as the visibility management makes all links light grey where one node leaves the context . Since the background is light grey and not white, we know that the continuous colour represents links.

7.2 Comparison with Related Work

Since this work is a follow-up of BiCFlows by Steinböck [Ste18], we decided to compare our implementation to their approach. As they also used the Media Transparency Database, it is easier to make direct comparisons based on scenarios. First, a summary of the similarities and dissimilarities of the frameworks is given, and then it is discussed how different scenarios are achieved with these frameworks.

The main similarities are that both works are focused on the support of free exploration. The frameworks should be able to motivate users to further investigate the presented

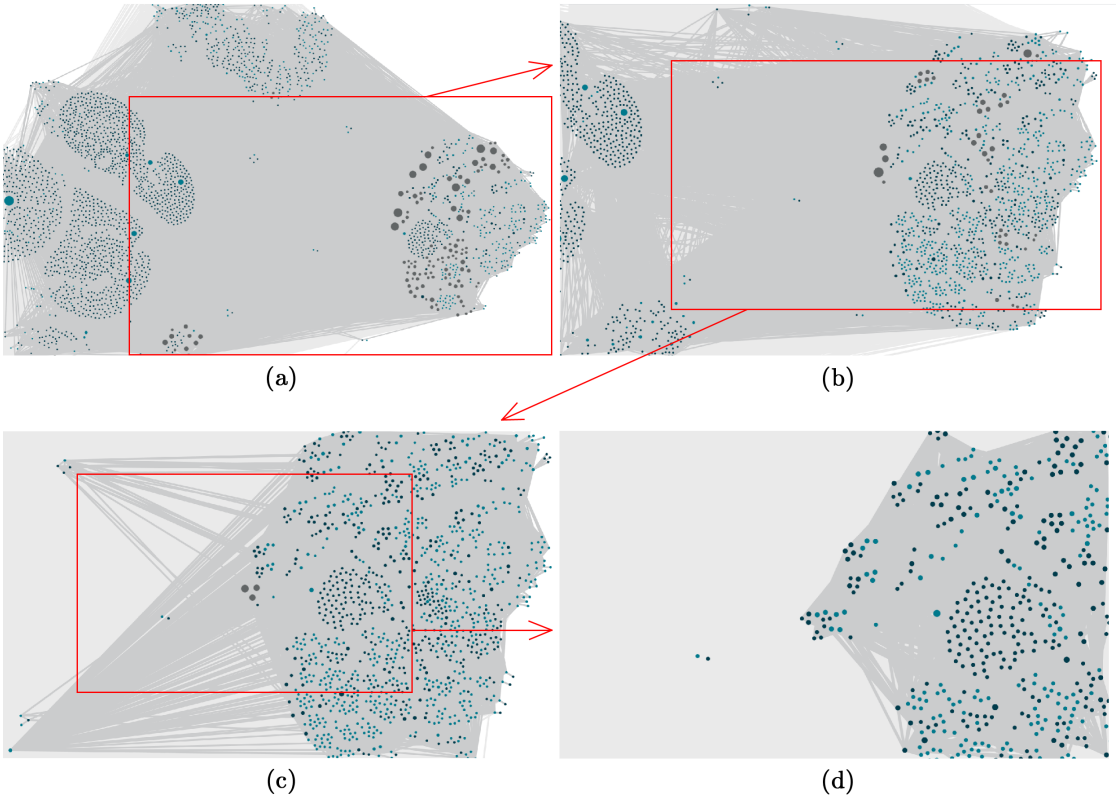


Figure 7.13: Views at levels six to nine of the medical dataset using seamless zoom. The parts which are zoomed in the next level are illustrated as red rectangles.

data. Another main common ground are the data characteristics. Both approaches are focused on large, weighted, bipartite datasets. To provide scalability due to the data size, both approaches use hierarchical aggregation and both frameworks use the coarsest level of the graph hierarchy as the initial view for the user. Regarding interactions, both frameworks apply highlighting on hover and on click the highlights are fixed. They also provide the possibility to filter for attributes. The interaction to drill-down the hierarchy, however, is different. BiCFlows shows the next level of a selected cluster by double-clicking it. The approach presented in this thesis provides seamless zoom and the firework animations to drill-down and keep context. Additionally, the context pull was implemented to bring the direct neighbours of a selected node into the context. In both frameworks, it is possible to highlight an entity by its name through the hierarchy, but our approach also implements such a highlighting by the specified attributes.

For clustering, they use a biclustering algorithm, in contrast, our approach uses agglomerative clustering. This is to make sure that for the layout and the path-preserving hierarchy only connected nodes are within the same cluster. However, the main difference is the visual encoding. Steinböck [Ste18] uses a parallel list visualization where each side

represents the entries of one set as can be seen in Figure 7.14. An advantage of their approach is that they can directly show labels, whereas in our approach the labels are only provided on-demand. An advantage of our approach is that the topology of clusters is much better visible. For example, the isolated components, which we discussed in the first use case, are very hard to find in their approach.

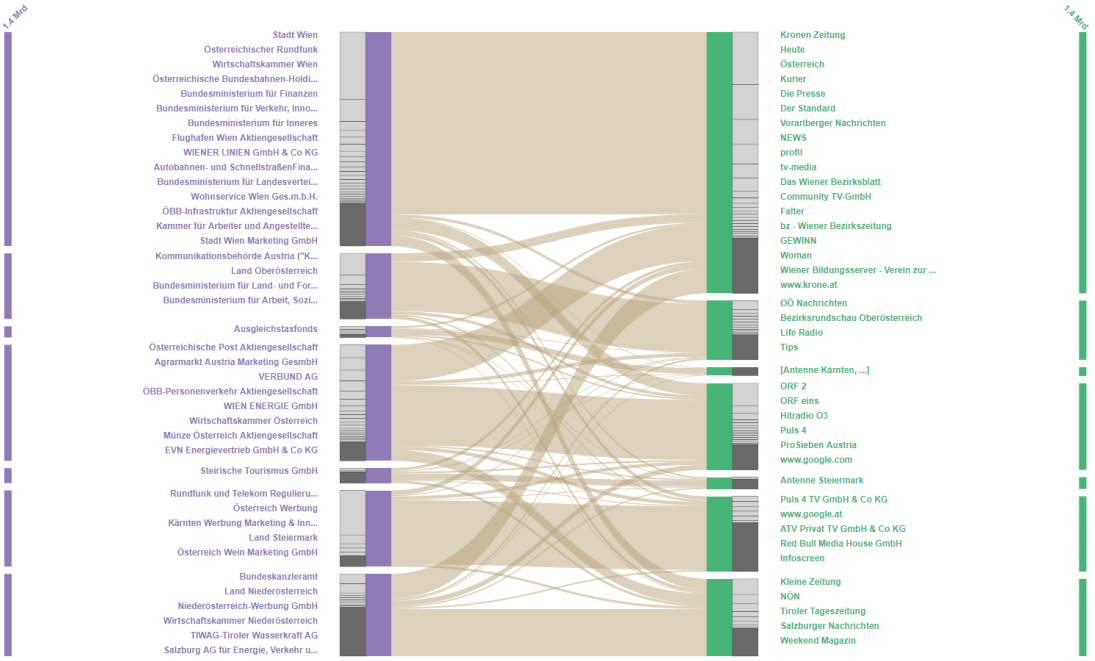


Figure 7.14: Visualization of the Media Transparency Database in BiCFlows [Ste18].

In their approach drilling-down the hierarchy is achieved only cluster-wise. Only the direct neighbours of the current clusters are visible and the user might lose track of the data. This may be one of the reasons why BiCFlows is perceived to be complex, as was one of the results in the user study by Steinböck. To overcome this issue we decided to use a different type of visualization that is known to be perceived as intuitive (see Chapter 4).

Apart from the differences described above, whether a user can fulfil the same tasks as he or she can by using BiCFlows is briefly discussed in the following. To do so, scenarios described in the use cases in the thesis by Steinböck are taken:

A user is interested in a specific entity. By drilling-down the hierarchy the user also discovers other unknown entities, which are connected to it. In BiCFlows, if the label is not already visible in the initial view, the user can select an entity in a list and then follow the highlighting through the hierarchy to the last level. There the user sees all relations of this entity. The listed labels now may indicate unknown entities. In our approach, the user defines a highlighting based on the label and also follows the highlighting through the hierarchy. The user can now explore the connected data nodes to retrieve their names.

Similar to the previous scenario is the situation where a user wants to find all entities of the other set that are connected to a specified entity. In both frameworks, the user has to completely drill down the hierarchy to the last level to be sure to obtain all related data entities and not just relations to other clusters.

The last scenario describes a user that wants to find the co-membership of entities. By clustering the data, we already group nodes, which are highly related. In both approaches, exploring the entries of one set within a cluster gives a good impression about the co-memberships.

7.3 Benchmarks

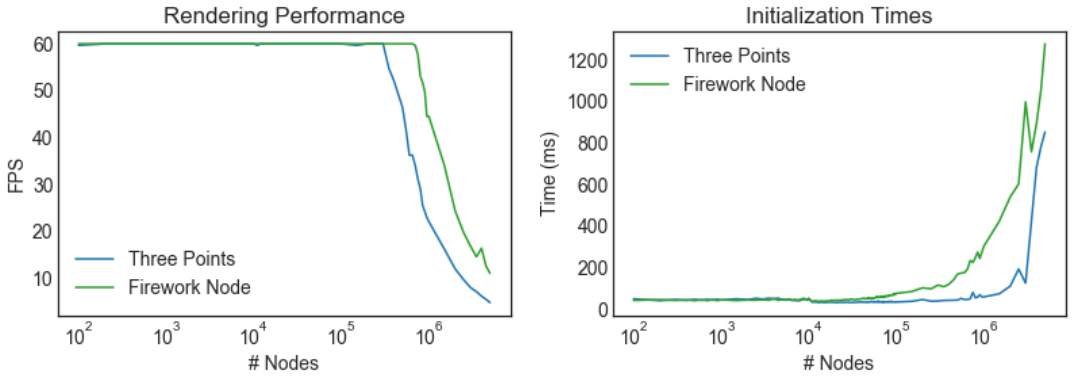
In Chapter 3 we showed that the average consumer hardware already struggles to render a graph with more than 6000 links. In this section, we show the results of different benchmarks that depict how much our implementation of the concept of Firework Plots could improve this interactive scalability issue. We also show benchmarks regarding our framework. In the following paragraphs, we will distinguish between two types of benchmarks. First, the implementations of our `Firework Node` and `Firework Link` shader are analyzed and discussed in terms of rendering performance. The second part covers the measurements of our framework regarding the preprocessing and initialization times as well as the implementation of the Firework Plots.

All tests, including the preprocessing steps on the server, were executed on a 64-bit Windows 10 machine with an Intel Core i5-6300U processor with 8 GB RAM and the built-in HD Graphics 520 graphics card with 4 GB memory. For the client-side benchmarks in the browser, we used Chrome with the 64-bit version 79.0.3945.130.

7.3.1 Firework Shader

In Chapter 6 we discussed our decisions to define a custom shader material. One reason was to be able to define the sizes of nodes and links per vertex instead of per buffer. Another reason was to calculate the firework animations on the GPU instead of expensive buffer manipulations on the CPU. The benchmarks in Chapter 3 already showed that the workaround, `Three Line2` [Lin], to render lines with linewidth other than one results in slightly worse performance. We wanted to test our implementation to see if the same performance as with the `Three Line2` implementation can be achieved. To do so, we use two different kind of scenes, similar to the benchmarks in Chapter 3. In one scene we test the rendering of nodes, starting with 10^2 up to more than 10^6 nodes. In the other test scene we render a graph with a fixed number of 4500 nodes and with 10^2 up to 10^5 links between those nodes. We measure the rendering performance in terms of FPS by collecting 100 frames per test scene, removing the first ten frames and calculating the average across the remaining 90 frames. The benchmarks were executed on the consumer hardware which was described in the previous paragraph.

In Figure 7.15, the results of rendering between 10^2 and more than 10^6 nodes are shown. Figure 7.15a shows the rendering performance in terms of FPS. It can be seen that our implementation has a slightly better rendering performance than the implementation by Three.js. This is because many of the calculations, as compared to the Three.js implementation, could be removed since the shader was fitted to the specific needs. For example, we do not need a support for dashed lines. Our implementation involves fewer calculations, but also a lot of functionality, like rendering nodes in different sizes and firework animations, were added. In terms of initialization times, the drawbacks can be seen in Figure 7.15b. The slower initialization times are because the firework shader needs more buffer attributes for the node sizes and the morph positions. At the initialization step, more buffers are created, which requires more time.



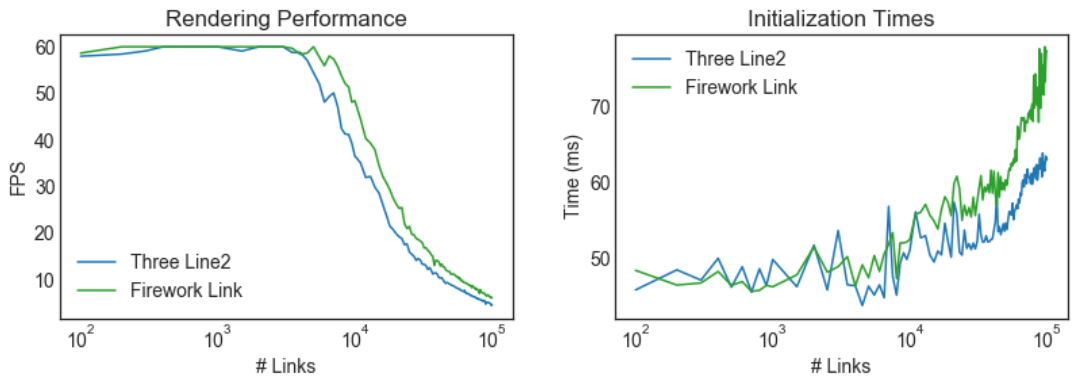
(a) (b)

Figure 7.15: Comparison of rendering performance of the Three.js material and our custom material in terms of FPS (a) and initialization times in milliseconds (b) with an increasing number of nodes with static positions.

Figure 7.16 shows the performance of rendering 10^2 to 10^5 links between 4500 nodes. The plots show very similar results to those rendering only nodes. Our implementation performs slightly better in terms of FPS (Figure 7.16a) with higher initialization times (Figure 7.16b). For example, for 10^4 links our `Firework Link` shader achieves ~ 48 FPS and the `Three Line2` implementation can only achieve up to ~ 36 FPS. The reasons are the same as for the node rendering. Additionally to custom linewidth and morph animations, our link shader program also contains the logic for the visibility management step of the firework plots.

7.3.2 Framework

In this subsection, the focus is on the performance of the framework itself. In contrast to the previous benchmarks, all results that are shown in the following paragraphs were recorded within the framework as they cannot be isolated from it. One part of this section comprises benchmarks showing the calculation times of the framework, including



(a) (b)

Figure 7.16: Comparison of rendering performance of the Three.js material and our custom material in terms of FPS (a) and initialization times in milliseconds (b) with an increasing number of links with static positions.

preprocessing and initialization times, as well as the layout calculation times. The other, more important part is to show the influence of the implementation of the Firework Plots on the rendering performance.

Setup

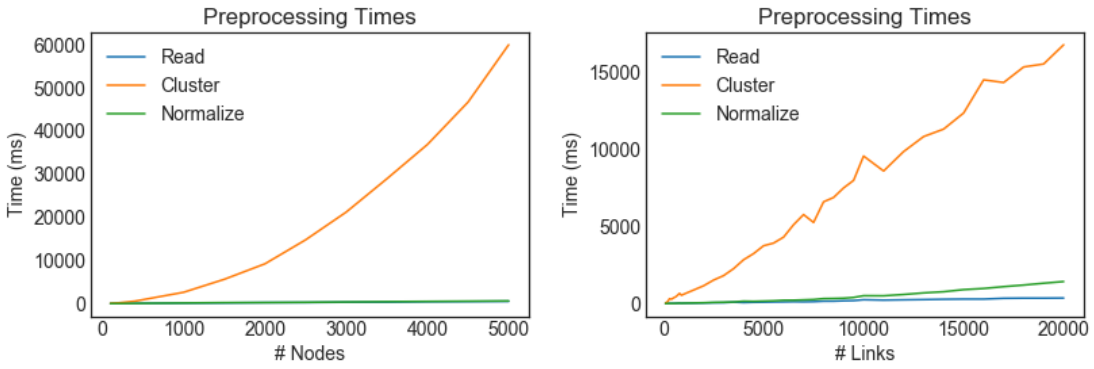
For the measurements in the next paragraphs, graphs were generated. To do so, the `bipartite.gnmk_random_graph()` function from the NetworkX [HSS08] library to generate random bipartite graphs was used. We created one type of graphs where we fixed the number of nodes at 512 nodes for each graph and the number of links was increased beginning with 100 links up to 20000. However, nodes, which are not incident to any link, are discarded. This not just leads to graphs of different sizes but also of different densities. In terms of linear density $D = \frac{|E|}{|V|}$ the smallest, sparsest generated graph has a density of ~ 0.55 and the largest, most dense graph has a density of ~ 19.53. According to Yoghourdjian et al. [YAD+19] the graphs are in the range of tree-like ($D < 1$) to very dense ($D > 4$). Additionally, we created test graphs where the density stays constant at $D = 4$. We started with 50 nodes per set and increased them up to 2500 nodes per set. We increased the number of links with the number of nodes such that the graph has a linear density of $D = 4$. At 2500 nodes in each set, this are 20000 links.

For the time measurements regarding the preprocessing and the layered graph layout, 30 runs per dataset were executed. From these values, the first five runs were removed and the average of the remaining 25 runs was calculated.

Preprocessing

The first conducted benchmarks regarding the framework were the preprocessing times. These are the only time measurements that were collected on the server. Also the server was executed on the previously described consumer machine.

Figure 7.17 depicts plots of the preprocessing times regarding both of the previously described graph setups. The preprocessing steps consist of reading the data, clustering it, and normalizing the edge weights. Measurements for each of these steps are shown in the plots. In Figure 7.17a the measurements of graphs with an increasing number of nodes and links and constant density can be seen. Figure 7.17b shows the results of the graphs with a constant number of nodes and an increasing number of links and, therefore, density. In both setups, the clustering step needed by far the most amount of time, as was to be expected. It can be seen that, at 5000 nodes in Figure 7.17a, which corresponds to 20000 links, the preprocessing times are around 60 seconds. In contrast, in Figure 7.17b, at 20000 links $\sim$ 15 seconds were measured. We can infer that the clustering time depends more on the number of nodes than on the number of links and the density. The preprocessing step is only computed once for a dataset, and the results are stored on the server.



(a) (b)

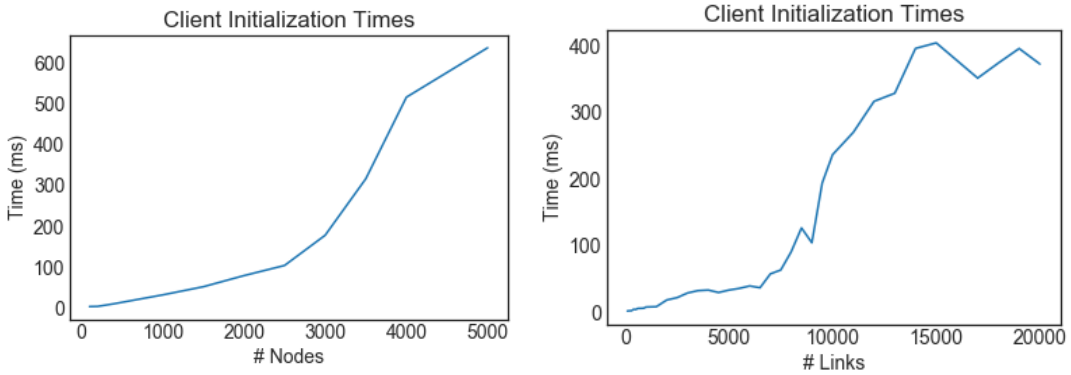
Figure 7.17: Plots showing the preprocessing times, consisting of times to read, cluster, and normalize the dataset. (a) shows measurements of graphs with an increasing number of nodes and links but a constant linear density and (b) illustrates the measurements of graphs with a fixed number of nodes, and an increasing number of links and, therefore, increasing linear density.

Initialization

Another test suite was for the measurement of the initialization times on the client-side. This concerns the time after a user received the requested data from the server until the rendering begins and the user can start the exploration. This is the time to load the data

and process it such that it can be properly used in the framework. Moreover, these are the latency times a user has to wait every time he or she starts our framework.

In Figure 7.18 the results of both graph setups are shown. It can be seen that the setup with an increasing number of nodes has longer initialization times. Note that these plots cannot directly be compared, as the initialization step depends on the number of elements. For example, at the last measurement of both setups, both graphs consist of 20000 links, but with a constant density (Figure 7.18a) there are 5000 nodes in the graph and in the other setup (Figure 7.18b) we have a fixed number of 1024 nodes (512 per set).



(a) (b)

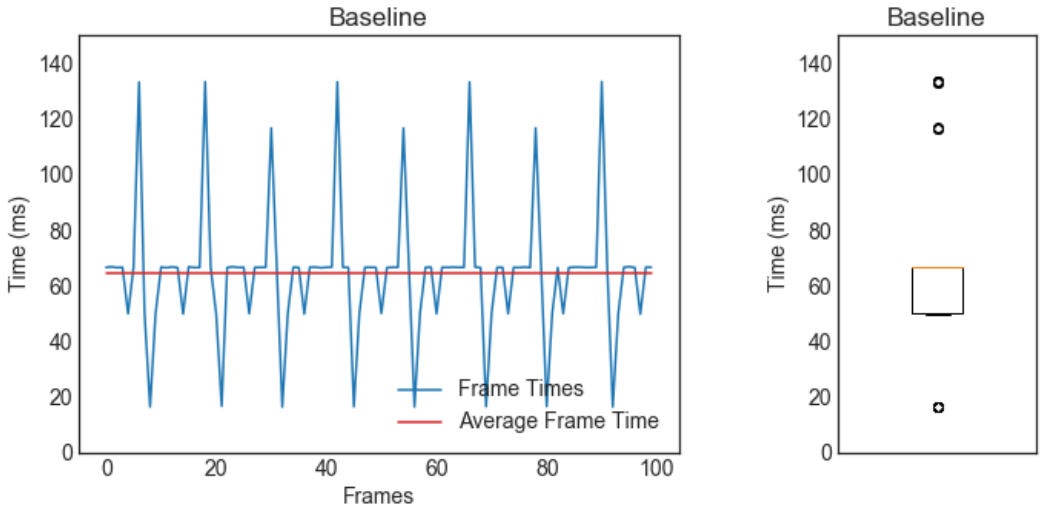
Figure 7.18: Plots showing the initialization times on the client-side to process the dataset received from the server before rendering it. (a) illustrates measurements of graphs with an increasing number of nodes and links but a constant linear density. (b) shows the measurements of graphs with a fixed number of nodes, and an increasing number of links and, therefore, increasing linear density.

7.3.3 Firework Plots

In this subsection, we aim to show the influence of our implementation of the concept of Firework Plots, presented in Chapter 5, on the scalability of rendering large graphs. To do so, we use the Media Transparency Database [Ope], which was also employed in one of the use cases. Following from the web-based graph rendering analysis in Chapter 3, it is obvious that this graph with 17780 links already runs into scalability issues on average consumer hardware. How we handle visual clutter, was shown with the use cases. How much Firework Plots improve the rendering performance, will be shown in the next paragraphs. To do so, we measure the frame times, rendering the whole dataset without any of the introduced strategies to introduce a baseline. Then we will add hierarchical aggregation to the scene and measure the frame times at each level. In the last step, we use the whole implementation of the Firework Plots, which were described in Chapter 5 and Chapter 6.

Baseline

For the baseline, we render the whole graph to the canvas without any of our introduced strategies. We collect the frame times of 100 frames. In Figure 7.19a the frame time of each of the 100 frames is plotted. We can see that the frame times are jumping up and down, which is due to the synchronization with the displays' refresh rate and not an issue with our framework. This was described in more detail in Chapter 3. The straight line shows the average across the 100 frames. Figure 7.19b shows a boxplot of the frame times which are shown in Figure 7.19a. We can see that the spikes are treated as outliers and that the mean frame time is ~ 68 ms which corresponds to four frames $17 \times 4 = 68$. The frame times of ~ 68 correspond to a rendering performance in terms of FPS of ~ 15 FPS.

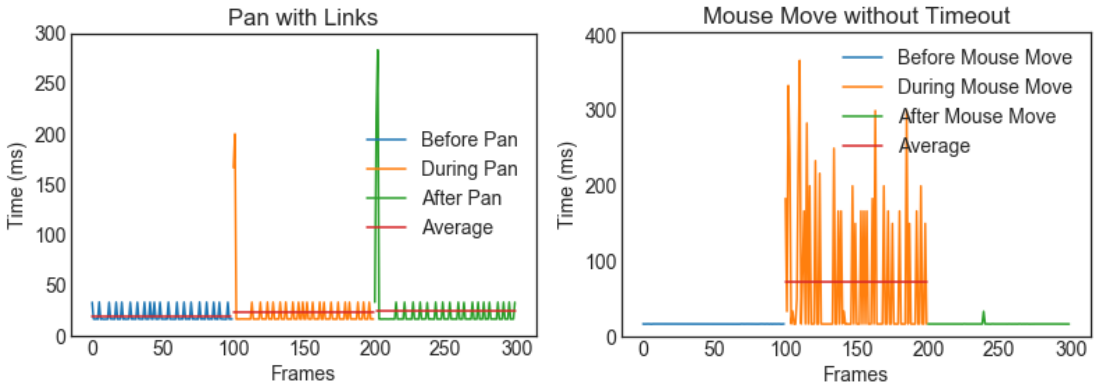


(a) (b)

Figure 7.19: In (a) the frame times of each frame are plotted, as well as the average frame time across these frames of the baseline setup. (b) shows a boxplot of of the collected frame times that are drawn in (a).

Since we also implemented improved functionality regarding the panning and mouse move interactions, we also show the frame times of these interactions, without any improvements, as part of the baseline. For these measurements, we selected specific use cases to show the influence of our implementation. For the pan interaction, a view that renders at around 50 FPS (~ 20 ms per frame) without any interaction was used. For the mouse move interaction, we used a view that renders at 60 FPS (~ 17 ms per frame). Figure 7.20 shows the frame times of the interactions before, during, and after the respective interaction, encoded in different colours. In these plots, we also added a line that depicts the average frame time across unstable measurements, for better comparison with the following plots. During the pan interaction (Figure 7.20a) we can

see no difference at all. Note that the two spikes in Figure 7.20a are because the mouse moves on the canvas to start and end the pan interaction also trigger raycasts. While moving the mouse over the canvas, many expensive raycasts are triggered, which result in slow rendering, as can be seen in Figure 7.20b.



(a) (b)

Figure 7.20: Frame times before, during and after pan (a) and mouse move (b) interaction. For unstable measurements also the average is plotted in red.

Hierarchical Aggregation

As a first step, we add the hierarchical aggregation, as described in Chapter 5, to the framework. We collected the frame times of 100 frames for each level. The whole graph at each aggregation level was rendered to the canvas. We expected the frame times to get slower with each aggregation level and to reach the same performance as the baseline with the last level. In Figure 7.21 the frame times are plotted using boxplots. The first two aggregated levels can be rendered without any performance issues. When more elements are added to the scene, the frame times are increasing. This can be seen starting from the third level. The mean of the third level is at $\sim$ 34 ms, which corresponds to two display refreshes. This means to redraw the scene, two display refreshes are needed. At the fourth level, the mean is at $\sim$ 51 ms, i.e., three display refreshes. Level 5 and level 6 of the graph hierarchy have approximately the same number of elements in the scene (see Table 7.1). This is also visible in the boxplots, as those of level 5 and level 6 are identical. Level 6 of the graph hierarchy shows the same scene as the baseline, hence the identical boxplots make sense.

By introducing hierarchical aggregation, we also have to mention the performance of our layered force-based graph layout. This is an expensive part of the framework, which is executed on the client-side. Measuring the calculation times and discussing them is important. To do so, a setup as before was used where each level of each graph was calculated 30 times and the first five measurements were removed to calculate the average.

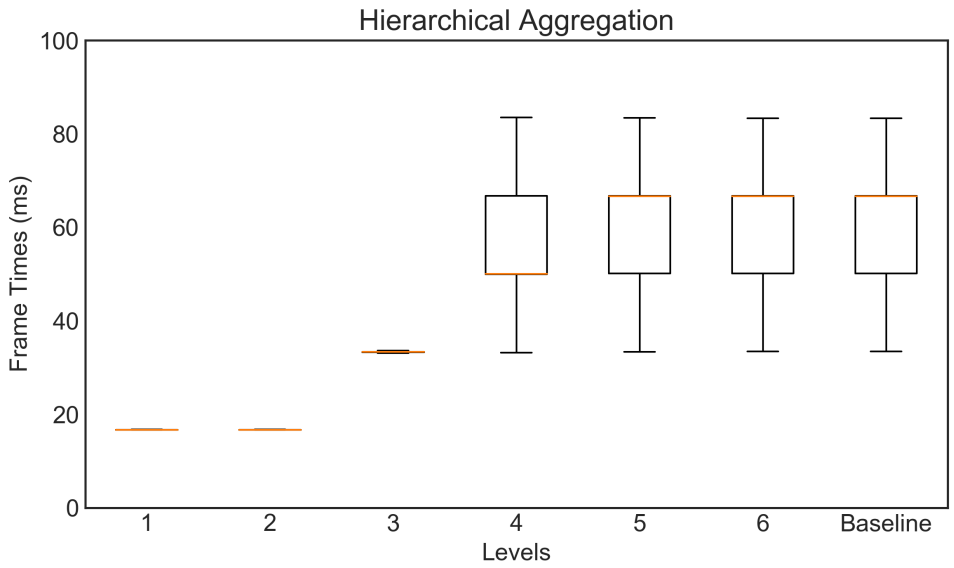
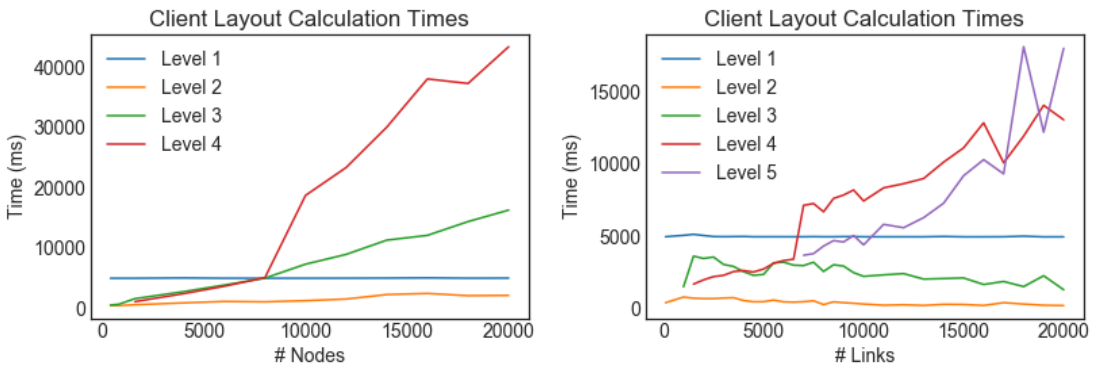


Figure 7.21: Boxplots of the frame times at each level of the graph hierarchy and the baseline. With more detail, rendering needs more time.

Figure 7.22 shows the measurements for different graph sizes. Figure 7.22a depicts the setup where the number of nodes and links in the graph are increased such that the linear density of the graphs is always $D = 4$. In Figure 7.22b, graphs with a fixed number of nodes, and an increasing number of links, and increasing density, are used.

In both plots, it can be seen that the calculation of the first level is constant, independent of the graph size, but still very expensive regarding the small graph size that can be expected from the first aggregation level. With every tick of the simulation the positions in the buffers are updated and rendered to the canvas. The ticks of the simulation are now also synced with the display's refresh rate which results in this stable calculation times. This is done because the user should be able to already start the observation and exploration as soon as possible. All the other level simulations are calculated in a worker thread, and therefore do not block the main thread the user can already interact with. With more levels, more nodes and links are added to the simulation and therefore the calculation gets more expensive. We can also see that the calculation times are higher if there are more nodes in the scene. One reason could be that the collision avoidance is only defined on the nodes and more nodes also require more computations. The simulation times depend on the parameters, which we defined as described in Chapter 6, and therefore, these measurements only provide a rough estimate of the expected times in the framework. As the calculations are so expensive they are done only once and can then be requested from the server afterwards.



(a) (b)

Figure 7.22: Plots of the layered graph layout calculation times. (a) depicts the setup where the number of nodes and links in the graph are increased such that the linear density of the graphs is always $D = 4$. In (b), graphs with a fixed number of nodes, and an increasing number of links, and increasing density, are used

Firework Plots

For the last step, we used the whole implementation of the Firework Plots concept as described in Chapter 5. This means, in the following measurements hierarchical aggregation, seamless zoom, and visibility management was used. In Figure 7.23 the measurements for all levels of one use case are given using boxplots.

Compared to hierarchical aggregation on its own in Figure 7.21, it can be seen that the first two levels are identical. Starting at level three the performance improvements using Firework Plots are visible. The boxplots of the last levels indicate that the frame times can be halved using our introduced concepts.

Although the rendering performance is better, we do not reach a constant frame rate throughout all levels as we hoped. This is because when using seamless zoom, generally more of the screen space that is rendered into is covered with scene elements. Therefore, more fragments are emitted by the rasterization process. For each fragment, the depth test is performed and the fragment shader is executed. Moreover, due to the density of the graphs, there are many overlapping lines. As these lines are rendered at the same z-level, depth testing does not remove any of those lines, which leads to a lot of overdrawing. The same pixel is redrawn multiple times within one draw call. This also means that the frame times depend on where the user zooms to. If she or he zooms to the small connected components, the frame rate will be stable throughout the hierarchy, as a small connected component also only requires a small portion of the screen space. For future work, a more sophisticated approach that tries to avoid overdrawing should be considered.

Since the scene is mostly static, the user only notices the slow performance during

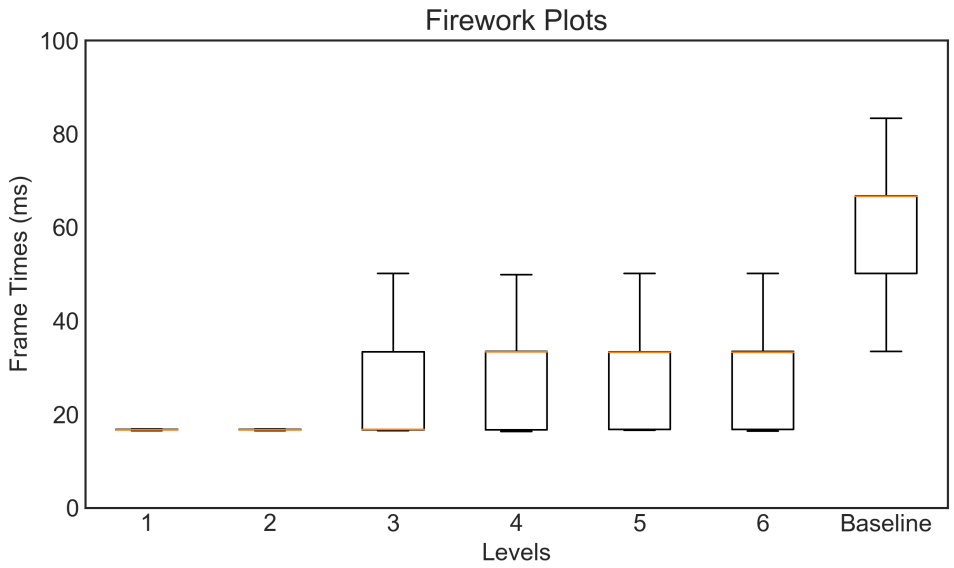
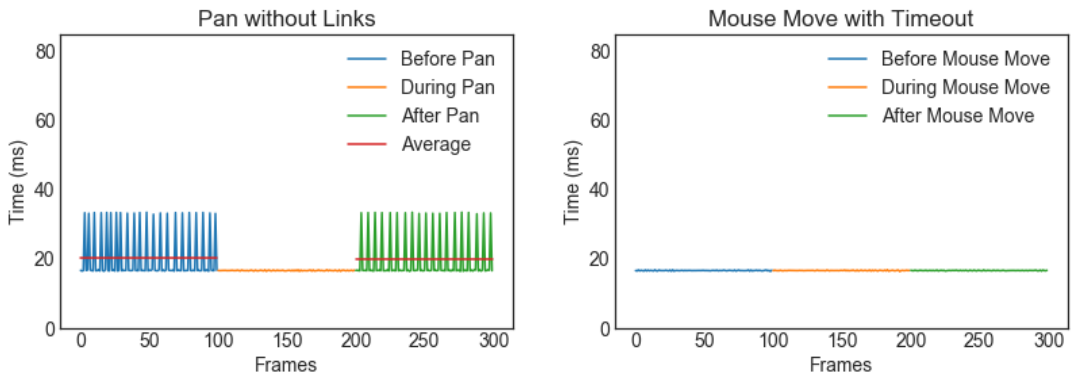


Figure 7.23: Boxplots of the frame times at all levels using Firework Plots and the baseline.

interactions. Therefore, we introduced improvements regarding the pan and mouse move implementation in Chapter 6. The influence of these strategies on the performance during user input can be seen in Figure 7.24. In the baseline of panning in Figure 7.20 it was visible that the panning itself does not influence the performance at all. However, when a scene renders at 50 FPS, also this interaction is done at 50 FPS. Therefore, to improve the rendering performance during panning itself, lines are hidden. The improvement, compared to the baseline, can be seen in Figure 7.24a. Before and after the user input the average frame times are $\sim$ 22 ms, and during the interaction, the frame times are stable at $\sim$ 17 ms.

Raycasting the scene is expensive, as can be seen in the baseline of the mouse move interaction in Figure 7.20b. We need to raycast the scene for on-demand labelling of the scene elements. In the baseline, it is obvious that this interaction needs improvements and cannot be triggered all the time during mouse move. In Figure 7.24b the improvements using a timeout as described in Chapter 6 can be seen. A timeout is used that only triggers the raycast if the mouse position stays at the same position for a certain time. The performance stays stable at $\sim$ 17 ms, because no raycasts are triggered when a user moves the mouse over the screen.



(a) (b)

Figure 7.24: Frame times before, during and after pan (a) and mouse move (b) interaction with improvements. For unstable measurements also the average is plotted in red.

7.4 Discussion

This chapter aimed to show how our approach handles the two main challenges of perception and interaction that arise when visualizing large datasets. Intuitive visualizations and interactions were chosen and combined to yield a framework for lay users.

To provide an intuitive visualization for inexperienced users, we decided to use node-link diagrams, as suggested by multiple user studies (see Chapter 6). To improve scalability hierarchical aggregation is used. The hierarchical aggregation adds a lot of elements to the visualization as was depicted in the use cases. Also, the denser the relations in a dataset are, the denser are the aggregated graph levels. But, the hierarchical clustering is important to provide an overview of the data. Also, it helps to orientate and keep context. The use cases showed that semantic clusters can be found easily and that their relations can be investigated.

For the graph hierarchy, we decided to use a force-based layout for each level, because it emphasizes the structure of the data very well. User studies showed that it is preferred by users (see Chapter 6). The calculation of the layered graph layout is very expensive. This lead to the implementation using web workers and the possibility to start exploration form the very first level even though the other levels might not be calculated yet. The layout calculation is only done once and the results can be retrieved from the server afterwards. Still, considering other, less expensive, graph layout algorithms and moving the calculation to the backend as part of the preprocessing should be considered as one of the next steps. We did not consider edge-bundling because some user studies state that curved edges are perceived as more complex for inexperienced users than straight edges. Also pathfinding tasks are harder to achieve using curved, bundled, edges instead of straight, single, ones (see Chapter 4). As we already have a hierarchical aggregation, hierarchical edge aggregation could be implemented easily in a preprocessing step. It

is an interesting task for future work to determine whether edge-bundling improves the visual encoding enough to compensate the drawback.

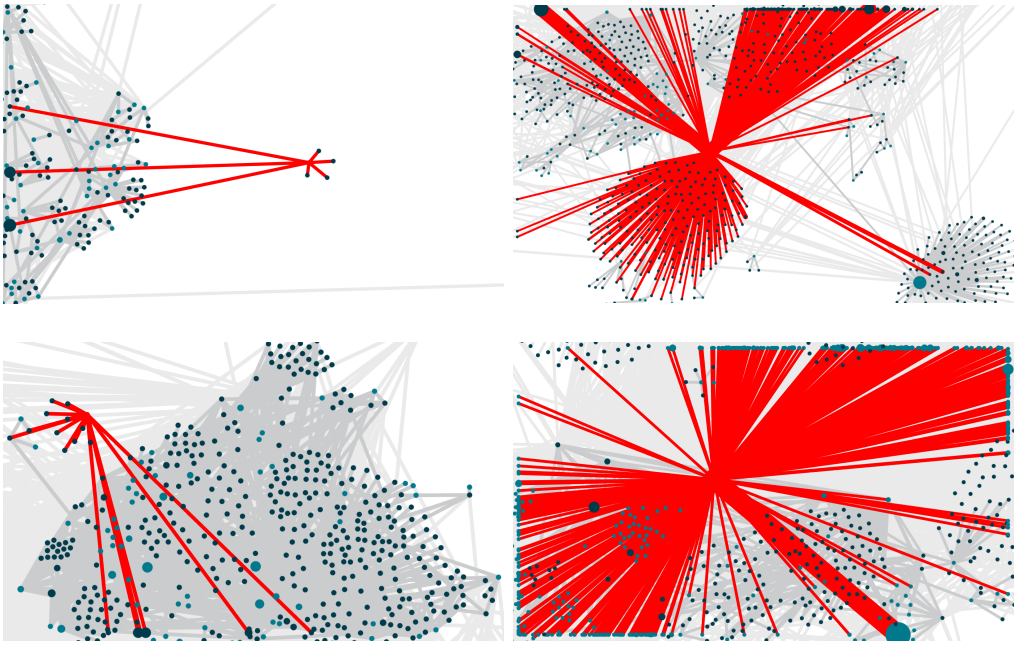
The hierarchical aggregation alone does not solve the problem of scalability as our benchmarks showed. Seamless zoom is introduced drilling-down the hierarchy. As zooming itself is a common interaction technique, we decided to couple it with drilling-down such that this happens without any additional user input. By going deeper into the hierarchy, this interaction technique leads to rendering smaller contexts with more details. Visibility management arranges nodes of a cluster, which spatially closer, into the same buffers to achieve better frustum culling. It also discards the rendering of links whose nodes are not visible. With these two functionalities, we aimed to improve rendering performance to yield a constant frame rate when drilling-down the hierarchy. Our benchmarks show an improvement compared to plain hierarchical aggregation, however, we could not completely reach the expectations. This is due to the large number of overlapping links, which leads to many redraws of the same pixel. Because of the seamless zoom, the deeper it is zoomed into the hierarchy, the more screen space is covered, and therefore more fragment calculations are needed.

One problem with seamless zoom is that links to nodes outside of the context can not be seen anymore. To overcome this we implemented the context pull, which pulls the neighbours of a node into the current context. It only works well, as long as the number of neighbours outside the context is small. In this approach, the nodes are pulled into the context along their links. This might lead to overlaps and clutter. An implementation to get a better positioning of the nodes inside the context needs to be considered. Figure 7.25 shows some examples of good and bad context pulls.

We tried to improve the rendering performance during user interactions as this is the moment when slow rendering becomes obvious. By omitting links during the pan interaction and using a timeout that triggers the raycasts for on-demand labelling, the rendering performance, as well as the user experience is improved. Labelling in a node-link diagram is a hard task. Labels which are always visible would add even more clutter. Therefore, labels are only shown on-demand. Labelling from the start could immediately support the user's orientation if he or she sees known entities from the beginning. More elaborate labelling could help to improve the user's orientation.

Highlighting and filtering is implemented to support attribute-based graph analysis tasks. The propagation of highlighting and filtering motivates the exploration of the data through the hierarchy to find relations and co-memberships with specified attributes.

The use cases indicate that the specific overview task of estimating the size of a graph is not as easy to execute due to the clustered view and the combination with seamless zoom. However, the focus in this thesis was not on high-level tasks as, for example, the comparison of two networks. The possibilities of the framework and how the user is motivated to explore the clustering and layout through the graph hierarchy are described in the user cases. They also illustrate how the Firework Plots can improve the visual



(a) (b)

Figure 7.25: Some examples of good (a) and bad (b) context pulls.

clutter without the user losing the context. We showed how most of the common graph analysis tasks as described by Lee et al. [LPP+06] can be completed in our framework.

CHAPTER 8

Conclusion and Future Work

In this thesis, we developed an interactive web-based framework that uses well-known visualization concepts and common interaction techniques to motivate the exploration of large bipartite datasets for lay users.

We analyzed existing web-based rendering techniques based on their performance and then further investigated existing libraries to obtain the best technique and framework in rendering as many elements as possible. We investigated the current limitations on a high-end consumer machine and found that, on average consumer hardware, the rendering already starts to slow down for > 6000 links.

To overcome the visual clutter and rendering issue that arises with large graphs, we introduced Firework Plots, a combination of visualization and interaction concepts for large graphs. Firework Plots make use of the well-known visualization technique of node-link diagrams and combines them with hierarchical aggregation to obtain better scalability and support user orientation. A layered, constrained, force-based graph layout is used due to the good readability of force-based layouts and to register the different levels of the graph hierarchy with each other. Seamless zooming aims to naturally drill-down the graph hierarchy as well as to improve interactive scalability by rendering smaller contexts. Animations are added for the level switches, which help to follow entities through the graph hierarchy and their cluster memberships. To further improve perceptual as well as interactive scalability, we introduced a visibility management. This aims to discard unnecessary links within a context as well as support frustum culling by putting spatially close nodes and links in the same buffers. Based on the web-based graph rendering analysis, we implemented our framework and the concept of Firework Plots as well as an interaction system and attribute-based highlighting and filtering.

To illustrate the improvements of visual clutter, we described two different use cases. These use cases describe how different graph analysis tasks can be achieved with our framework and how the user is supported to explore the graph hierarchy. These use

cases also showed that our framework is not suitable to achieve high-level graph analysis tasks as, for example, estimating the number of nodes or comparing two networks. We conducted multiple benchmarks to measure the influence of the concept of Firework Plots on the rendering performance. These benchmarks indicate that we could improve on the scalability issues. We hoped to achieve constant frame rates during seamless zoom, which we could not entirely fulfil. The measurements also showed that especially the one-time calculations in our framework, as hierarchical clustering and layout calculation, are very expensive and should be improved.

Future Work

During the development of this thesis, we had many ideas regarding the improvement of the implementation and our concepts. Many of these ideas made it into the version, which we presented in this thesis. There are still a few ideas, which can be considered as future work.

One of the main tasks in every framework is to decrease calculation times. As already mentioned, considering different clustering algorithms, which have better a performance is important. As we already have hierarchical aggregation, implementing hierarchical edge-bundling could be one of the next steps to improve the rendering performance.

Force-directed layouts are known to have high calculation times, which was also shown in our benchmarks. There already exist many approaches, which improve these. Investigating such approaches and implementing one in the framework could help a lot. A more performant layout calculation also offers the possibility to recalculate a graph layout when filtering is applied. Moving the layout calculation to the backend also is a very important step, as long as no user interaction is required.

Another important part is to improve the user experience. To support orientation for a user a more elaborate labelling concept could be helpful. In the beginning, this could just mean to have fixed labels for the largest entities. It could be extended to have an automatic cluster labelling based on the data input. Implementing bipartite projections could help the user to explore the entity connections of one set and find co-memberships between them more easily. This could also be worth investigating in combination with the previously mentioned layout recalculation.

Also regarding the implementation itself, improvements should be made as next steps. The current implementation suffers from a lot of overdrawing because all links are at the same z-level. Depth testing does not discard any of those links which leads to many, unnecessary, fragment calculations of the same fragment. One idea for improvement here is to set the z-coordinate of links according to their weight, such that less important links are likely to be discarded by depth testing if there are more important links rendered in front of them.

Bibliography

[ADWM04] A. T. Adai, S. V. Date, S. Wieland, and E. M. Marcotte. Lgl: Creating a map of protein function with an algorithm for visualizing very large biological networks. *Journal of Molecular Biology*, 340(1):179–190, 2004.

[AHK06] J. Abello, F. V. Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):669–676, Sep 2006.

[AKY05] J. Abello, S. G. Kobourov, and R. Yusufov. Visualizing large graphs with compound-fisheye views and treemaps. In J. Pach, editor, *Graph Drawing*, pages 431–441, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[AMA07] D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, Mar 2007.

[AMA08] D. Archambault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):900–913, Jul 2008.

[AMA09] D. Archambault, T. Munzner, and D. Auber. TugGraph: Path-Preserving Hierarchies for Browsing Proximity and Paths in Graphs. In *IEEE Pacific Visualization Symposium*, pages 113–121, Pekin, China, Apr 2009.

[Av04] J. Abello and F. van Ham. Matrix zoom: A visual interface to semi-external graphs. In *IEEE Symposium on Information Visualization*, pages 183–190, Oct 2004.

[BBD+11] V. Batagelj, F. J. Brandenburg, W. Didimo, G. Liotta, P. Palladino, and M. Patrignani. Visual analysis of large graphs using (x,y)-clustering and hybrid visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1587–1598, Nov 2011.

[BBHR+16] M Behrisch, B. Bach, N. Henry Riche, T. Schreck, and J. Fekete. Matrix reordering methods for table and network visualization. *Computer Graphics Forum*, 35(3):693–716, 2016.

[BBP+06] S. Barkow, S. Bleuler, A. Prelić, P. Zimmermann, and E. Zitzler. BicAT: a biclustering analysis toolbox. *Bioinformatics*, 22(10):1282–1283, Mar 2006.

[BD07] M. Balzer and O. Deussen. Level-of-detail visualization of clustered graph layouts. In *2007 6th International Asia-Pacific Symposium on Visualization*, pages 133–140, Feb 2007.

[BDL05] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 165–172, New York, NY, USA, 2005. ACM.

[BKC10] D. Bozdağ, A. S. Kumar, and U. V. Catalyurek. Comparative analysis of biclustering algorithms. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, BCB'10, pages 265–274, New York, NY, USA, 2010. ACM.

[BKH05] F. Bendix, R. Kosara, and H. Hauser. Parallel sets: visual analysis of categorical data. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 133–140, 2005.

[BL07] R. Blanch and E. Lecolinet. Browsing zoomable treemaps: Structure-aware multi-scale navigation techniques. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1248–1253, Nov 2007.

[BOH11] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[Boo] Bootstrap. https://getbootstrap.com, accessed 11.03.2020.

[Can] HTML Canvas Tutorial. https://www.w3schools.com/graphics/canvas_intro.asp, accessed 25.04.2020.

[CGK+07] R. Chang, M. Ghoniem, R. Kosara, W. Ribarsky, J. Yang, E. Suma, C. Ziemkiewicz, D. Kern, and A. Sudjianto. Wirevis: Visualization of categorical, time-varying data from financial transactions. In *2007 IEEE Symposium on Visual Analytics Science and Technology*, pages 155–162, Oct 2007.

[CZQ+08] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, Nov 2008.

[d3-a] d3-scale. https://github.com/d3/d3-scale, accessed 03.04.2020.

[d3-b] d3-zoom. https://github.com/d3/d3-zoom, accessed 01.04.2020.

[Dan12] B. Danchilla. *Three.js Framework*, pages 173–203. Apress, Berkeley, CA, 2012.

[DLF+09] T. Dwyer, B. Lee, D. Fisher, K. I. Quinn, P. Isenberg, G. Robertson, and C. North. A comparison of user-generated and automatic graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):961–968, Nov 2009.

[EDG+08] N. Elmqvist, T. Do, H. Goodell, N. Henry, and J. Fekete. Zame: Interactive large-scale graph visualization. In *2008 IEEE Pacific Visualization Symposium*, pages 215–222, 2008.

[EF10] N. Elmqvist and J. Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454, May 2010.

[EHP+11] O. Ersoy, C. Hurter, F. Paulovich, G. Cantareiro, and A. Telea. Skeleton-based edge bundling for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2364–2373, Dec 2011.

[Fau] I. Fauzi. Typeahead. https://iqbalfn.github.io/bootstrap-typeahead, accessed 01.04.2020.

[FB95] G. W. Furnas and B. B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, page 234–241, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[FHK+09] D. C. Y. Fung, S. Hong, D. Koschützki, F. Schreiber, and K. Xu. Visual analysis of overlapping biological networks. In *2009 13th International Conference Information Visualisation*, pages 337–342, Jul 2009.

[For10] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[FR91] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, Nov 1991.

[FSB+13] P. Fiaux, M. Sun, L. Bradel, C. North, N. Ramakrishnan, and A. Endert. Bixplorer: Visual analytics with biclusters. *Computer*, 46(8):90–94, Aug 2013.

[Fur86] G. W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, pages 16–23, New York, NY, USA, 1986. ACM.

[GBMK08] J. Gardy, A. Barsky, T. Munzner, and R. Kincaid. Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE Transactions on Visualization and Computer Graphics*, 14(06):1253–1260, Nov 2008.

[GFC05] M. Ghoniem, J. Fekete, and P. Castagliola. On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, Jul 2005.

[GHGH09] A. Godiyal, J. Hoberock, M. Garland, and J. C. Hart. Rapid multipole graph drawing on the gpu. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing*, pages 90–101, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[GK07] E. R. Gansner and Y. Koren. Improved circular layouts. In M. Kaufmann and D. Wagner, editors, *Graph Drawing*, pages 386–398, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[GKL+13] S. Ghani, B. C. Kwon, S. Lee, J. S. Yi, and N. Elmqvist. Visual analytics for multimodal social network analysis: A design study with social scientists. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2032–2041, Dec 2013.

[Gri14] M. Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 1st edition, 2014.

[HB05] J. Heer and D. Boyd. Vizster: Visualizing online social networks. In *IEEE Information Visualization (InfoVis)*, pages 32–39, 2005.

[HFM07] N. Henry, J. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, Nov 2007.

[Hol06] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, Sep 2006.

[HSS08] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA, 2008.

[HvW09] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. In *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'09, page 983–998, Chichester, GBR, 2009. The Eurographs Association & John Wiley & Sons, Ltd.

[HW16] W. S. Hartono and D. H. Widyantoro. Fisheye zoom and semantic zoom on citation network visualization. In *2016 International Conference on Data and Software Engineering (ICoDSE)*, pages 1–6, Oct 2016.

[JP] A. Jacomy and G. Plique. Sigma.js. http://sigmajs.org, accessed 07.02.2020.

[jQu] jQuery. https://jquery.com, accessed 11.03.2020.

[KEC06] R. Keller, C. M. Eckert, and P. J. Clarkson. Matrices or node-link diagrams: Which visual representation is better for visualising connectivity models? *Information Visualization*, 5(1):62–76, 2006.

[KK89] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7 – 15, 1989.

[KM13] R. Kosara and J. Mackinlay. Storytelling: The next step for visualization. *Computer*, 46(5):44–50, May 2013.

[LBA10] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Computer Graphics Forum*, 29(3):853–862, 2010.

[LCL+09] L. Shi, N. Cao, S. Liu, W. Qian, L. Tan, G. Wang, J. Sun, and Y. Lin. Himap: Adaptive visualization of large-scale online social networks. In *2009 IEEE Pacific Visualization Symposium*, pages 41–48, Apr 2009.

[Lin] Line2.js. https://github.com/mrdoob/three.js/blob/dev/examples/jsm/lines/Line2.js, accessed 28.03.2020.

[LL19] J. Leech and B. Lipchak. OpenGL ES Version 3.0.6. Technical report, Khronos Group, 2019.

[LLCM12] S. Luo, C. Liu, B. Chen, and K. Ma. Ambiguity-free edge-bundling for interactive graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):810–821, May 2012.

[LPP+06] B. Lee, C. Plaisant, C. Parr, J. Fekete, and N. Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, page 1–5, New York, NY, USA, 2006. Association for Computing Machinery.

[McC] L. McCarthy. p5.js. https://p5js.org, accessed 28.03.2020.

[MD12] F. McGee and J. Dingliana. An empirical study on the impact of edge bundling on user comprehension of graphs. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 620–627, New York, NY, USA, 2012. ACM.

[Med] Bundesverfassungsgesetz über die Transparenz von Medienkooperationen sowie von Werbeaufträgen und Förderungen an Medieninhaber eines periodischen Mediums (Medienkooperations- und -förderungs-Transparenzgesetz, MedKF-TG). https://www.ris.bka.gv.at/eli/bgbl/I/2011/125/20111227, accessed 19.03.2019.

[Mis06] K. Misue. Drawing bipartite graphs as anchored maps. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60*, APVis '06, page 169–177, AUS, 2006. Australian Computer Society, Inc.

[MO04] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, Jan 2004.

[New04] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6), Jun 2004.

[Ope] Open Data – Medientransparenz Datenbekanntgabe. https://www.rtr.at/de/inf/odMTGDaten, accessed 19.03.2019.

[PC17] V. A. Padilha and R. J. G. B. Campello. A systematic comparative evaluation of biclustering techniques. *BMC Bioinformatics*, 18(1):55, Jan 2017.

[PFH+18] N. Pezzotti, J. Fekete, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Multiscale visualization and exploration of large bipartite graphs. *Computer Graphics Forum*, 37:549–560, Jun 2018.

[Poi] Points. https://threejs.org/docs/api/en/objects/Points, accessed 09.04.2020.

[Pop] Popper. https://popper.js.org, accessed 11.03.2020.

[PSD09] M. Pohl, M. Schmitt, and S. Diehl. Comparing the readability of graph layouts using eyetracking and task-oriented analysis. In *Proceedings of the Fifth Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Computational Aesthetics'09, pages 49–56, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[PZT+06] A. Prelić, E. Zitzler, L. Thiele, S. Bleuler, L. Hennig, P. Zimmermann, W. Gruissem, A. Wille, and P. Bühlmann. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122–1129, Feb 2006.

[QZW07] H. Qu, H. Zhou, and Y. Wu. Controllable and progressive edge clustering for large networks. In M. Kaufmann and D. Wagner, editors, *Graph Drawing*, pages 399–404, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[RLH17] D. Ren, B. Lee, and T. Höllerer. Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering. *Computer Graphics Forum*, 36(3):179–188, 2017.

[Sea] M. Seal. Agglomerative clustering tool for network-x graphs. https://pypi.org/project/AgglomCluster, accessed 31.03.2020.

[SGG+14] M. Streit, S. Gratzl, M. Gillhofer, A. Mayr, A. Mitterecker, and S. Hochreiter. Furby: fuzzy force-directed bicluster visualization. *BMC Bioinformatics*, 15(6):S4–S4, 2014.

[SGLS07] J. Stasko, C. Gorg, Z. Liu, and K. Singhal. Jigsaw: Supporting investigative analysis through interactive visualization. In *2007 IEEE Symposium on Visual Analytics Science and Technology*, pages 131–138, Oct 2007.

[SGW18] D. Steinböck, M. E. Gröller, and M. Waldner. Casual visual exploration of large bipartite graphs using hierarchical aggregation and filtering. In *2018 International Symposium on Big Data Visual and Immersive Analytics, BDVA*, pages 1–10, 2018.

[SMNR16] M. Sun, P. Mi, C. North, and N. Ramakrishnan. Biset: Semantic edge bundling with biclusters for sensemaking. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):310–319, Jan 2016.

[SRG+19] C. Stoiber, A. Rind, F. Grassinger, R. Gutounig, E. Goldgruber, M. Sedlmair, Š. Emrich, and W. Aigner. netflower: Dynamic network visualization for data journalists. *Computer Graphics Forum*, 38(3):699–711, 2019.

[Ste18] D. Steinböck. Interactive visual exploration interface for large bipartite networks. Master's thesis, Institute of Visual Computing & Human-Centered Technology, Vienna University of Technology, Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria, May 2018.

[STQ08] R. Santamaría, R. Therón, and L. Quintales. BicOverlapper: A tool for bicluster visualization. *Bioinformatics*, 24(9):1212–1213, Mar 2008.

[SVG] SVG Tutorial. https://www.w3schools.com/graphics/svg_intro.asp, accessed 25.04.2020.

[TAvS06] C. Tominski, J. Abello, F. van Ham, and H. Schumann. Fisheye tree views and lenses for graph visualization. In *Tenth International Conference on Information Visualisation (IV'06)*, pages 17–24, Jul 2006.

[TSS05] A. Tanay, R. Sharan, and R. Shamir. Biclustering algorithms: A survey. In *Handbook of Computational Molecular Biology Edited by: Aluru S. Chapman & Hall/CRC Computer and Information Science Series*, 2005.

[VMB+10] N. K. Verma, S. Meena, S. Bajpai, A. Singh, A. Nagrare, and Yan Cui. A comparison of biclustering algorithms. In *2010 International Conference on Systems in Medicine and Biology*, pages 90–97, Dec 2010.

[vR08] F. van Ham and B. Rogowitz. Perceptual organization in user-generated graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1333–1339, Nov 2008.

[Web] OpenGL ES for the Web. https://www.khronos.org/webgl, accessed 25.04.2020.

[XCQS16] P. Xu, N. Cao, H. Qu, and J. Stasko. Interactive visual co-cluster analysis of bipartite graphs. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 32–39, Apr 2016.

[XRPH12] K. Xu, C. Rooney, P. Passmore, and H. Ham. A user study on curved edges in graph visualisation. In P. Cox, B. Plimmer, and P. Rodgers, editors, *Diagrammatic Representation and Inference*, pages 306–308, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[XW05] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[YAD+19] V. Yoghourdjian, D. Archambault, S. Diehl, T. Dwyer, K. Klein, H. C. Purchase, and Y. Wu. Exploring the limits of complexity: A survey of empirical studies on graph visualisation. *Visual Informatics*, 2(4), Jan 2019.

[ZPYQ13] H. Zhou, Panpan Xu, X. Yuan, and H. Qu. Edge bundling in information visualization. *Tsinghua Science and Technology*, 18(2):145–156, Apr 2013.