FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Stream I/O - Eine Interaktive Visualisierung von Publikationsdaten

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Michael Mazurek
Matrikelnummer 1126483

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Dr. techn. Manuela Waldner, MSc

Wien, 17. Februar 2017

Michael Mazurek	Manuela Waldner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Stream I/O - An Interactive Visualization of Publication Data

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Michael Mazurek
Registration Number 1126483

to the Faculty of Informatics
at the TU Wien
Advisor: Dr. techn. Manuela Waldner, MSc

Vienna, 17th February, 2017

Michael Mazurek

Manuela Waldner

Erklärung zur Verfassung der Arbeit

Michael Mazurek

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, 17. Februar 2017

Michael Mazurek

Kurzfassung

Die Publikationsdatenbank des Instituts für Computergraphik und Algorithmen ist eine klassische Benutzeroberfläche, welche ermöglicht gezielt nach Veröffentlichungen zu suchen. Gezielte Suchen werden durch Textfelder und Dropdown-Boxen ermöglicht und das Ergebnis wird in einer Liste dargestellt. Während eine gezielte Suche einfach sein kann, bei der die gesuchte Veröffentlichung a-priori bekannt ist, wird es bei ungezielten Suchen schon schwieriger. Stream I/O ist eine Anwendung, welche die Benutzeroberfläche der Publikationsdatenbank erweitert um solche Probleme zu lösen. Weitere Probleme sind die fehlende Übersicht sowie fehlende Unterstützung zur Analyse und Datenexploration. Ergebnislisten sind suboptimal für solche Aufgaben. Das Ergänzen einer interaktiven Visualisierung löst diese Probleme in der neuen Anwendung. Die Publikationsdatenbank stellt eine Auswahl von Attributen zur Verfügung, die viel Information über die enthaltenen Veröffentlichungen enthalten. Mit der Streamgraph [BW08] Visualisierung werden mehrere Attribute auf einer Timeline abgebildet um deren Veränderungen im Verlauf zur Zeit darzustellen. Hierbei liegt der Fokus auf dem Darstellen der Relationen zwischen Werten, welche ein Attribut annehmen kann, während gleichzeitig die Summe vermittelt wird. Da der Streamgraph die Timeline Metapher verwendet wird auch hier der Fluss von links nach rechts als Zeitfluss interpretiert. Die Bewegung auf der Zeitachse vermittelt die Entwicklung der Publikationsattribute, während das Vorkommen eines Attributes durch die Breite einer Schicht signalisiert wird. Durch das Filtern der Datenbank aus mehreren Perspektiven erhalten Benutzer/innen Einsicht wie die Attribute zueinander in Relation stehen und wie sich die Daten im Bezug auf Interaktionen ändern. Eine Auswahl von mehreren Schichten kann auf einen Trend in den Publikationsdaten hinweisen. Darüber hinaus stellen die Schichten der Visualisierung die Daten als eine Einheit dar. Die Silhouette der Schichten kann Schlüsse über die beinhalteten Daten als Ganzes ermöglichen. Deshalb ist Stream I/O eine nützliche Anwendung die Nutzer/innen beim durchsuchen der Datenbank unterstützt und darüber hinaus zum explorativen Erkunden anregt.

Abstract

The publication database of the Institute of Computer Graphics and Algorithms can currently be queried by a simple UI which returns a list. Stream I/O, the application of this thesis, extends the interface to improve it in terms of overview, exploration and analysis support. To cope with these shortcommings a visualization is added to the user interface. As the publication database includes a lot of additional data attributes, a selection of attributes is used for the visualization to give further insight. By using the Streamgraph [BW08] visualization, the variations over time within attributes like authors, publication type and research areas are made visible. The focus of this visualization lies in showing individual attribute values while also conveying the sum. This relationship is depicted in a timeline, which allows a user to explore the past and current work of the institute as well as to find relationships and trends in the publications. As the visualization uses a timeline encoding, the directed flow from left to right is interpreted as the movement through time. It shows the evolution of different attributes, while the occurrence of a topic at a specific time is coded with the width of the layer at a specific point. Searching the database is enriched through multiple viewpoints which give the user insight how attributes relate in the underlying data and how the data is changing through his/her manipulation. Selections of colored layers within the graph can represent bigger trends and give insight into the data as a whole. The Stream I/O application invites users to interactively explore the publication database, while simultaneously gaining new insight through the visualization.

Contents

CHAPTER 1

Introduction

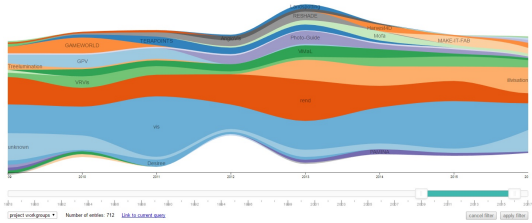
The publication database of the Institute of Computer Graphics and Algorithms currently provides the option to query publications based on specific attributes like the author, year or title. Through combination of these attributes any publication which is searched for can be found. Results are visualized in a list, which scales poorly with a big number of publications and is inefficient in showing relations in the data on its own. Beyond the lookup, no further assistance is given to analyze the database. Designing the interface in the way of presenting a list of publications is common practice. However, this approach misses an overview and the support for exploration without prior knowledge. Analytical methods like clustering are not supported and have to be done manually to get further insight. To improve the interface in respect to this issue, this thesis presents an user interface, that can be combined with the current interface. The goal of this project is to develop an interactive visualization to cope with the shortcomings of the publication database interface, by providing a visualization as well as additional interaction and analytical methods while staying simple.

To fulfill this goal, a visualization was chosen first, which defines a simple visual mapping that can be understood without much training and can provide overview and detail on demand. In the designing phase, several reasons lead to a time-dependent visualization, being the visualization of choice. In the current interface time is already the primary attribute, when the database is queried publications are sorted by time. Thus using a visualization that sorts the data the same way visually on a timeline fits nicely into the current interface. The timeline is a very intuitive visual mapping that is widely understood, and using this mapping allows the users to visually explore how publication activities have evolved over time. Also, the publication year attribute is one of the few attributes, that is available for almost all the data sets. All other attributes in the database are categorical (e.g, authors, keywords, workgroups, publication types). Therefore using the temporal attribute as the primary attribute allows for a quantitative model, where distances between elements can be measured.

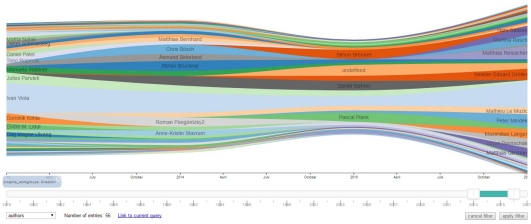
The visualization of choice is the Streamgraph by Byron et al. [BW08]. Rather than representing each publication individually, the Streamgraph is an aggregate visualization. It sums up the categorical values of a certain attribute and displays them over the time. This approach results in a value count of each value at each discrete time point. To emphasize the continuity of the values, they are interpolated between the time points. Thus a geometrical form for each value is derived, the layer. In the graph, time is mapped horizontally in a linear fashion from left to right. The amount of an certain value, in respect to the time, is shown by the vertical extent. Therefore individual layers hold much information about the underlying data as a whole in their geometry. Stacking the layers on top of each other shows the overall frequencies in the data and enables the comparison of attribute values at various points in time. There are plethora of other time dependent visualizations, however, this visualization was chosen because it is an intuitive and simple graph which can be explored without extensive instructions and is also appreciated for its aesthetics [BW08].

Further the application of this thesis, Stream I/O, is meant to be an interactive visualization that encourages analysis and exploration of the database. Exploration is enabled through filtering, changing between database attributes and adjusting the time span. As an example for exploration, a user could be interested in the work done in the research field of illustrative visualizations. At the start-up of the visualization, a default attribute (e.g., project workgroups) is shown without any filtering operations, the view is depicted in Figure 1.1(a). During the exploration of the database, the user is able to restrict the shown data with the mentioned tools, the Streamgraph visualization, as well as the list of publication, change accordingly to the restrictions. This can be seen in Figure 1.1(b): The user sets a filter onto illustrative visualization (IllVis) and changes to the author view to see if there is an author who is most involved in the publications of this workgroup. At this point the user searches for the biggest author layer, which is Ivan Viola. Then the user takes a look onto Ivan Violas participation in different workgroups, through setting a filter for him and removing the filter on the workgroups that was set earlier. In the graph (see Figure 1.1(c)) the user finds that Violas work focus shifted drastically to the IllVis workgroup, which fits the findings well. Finally a look is given onto the Streamgraph with selected from the view of the research area attribute, as the user is interested in how much of the work Ivan Viola has done in the workgroup that is also labeled with IllVis as research area. As each operation is saved, it is possible to restrict the shown data in relation to multiple attributes over a certain time period. So the user sets two filters, the first one on Ivan Viola and the second one on the IllVis workgroup and when the user switches to the research area graph (see Frigure1.1(d)), he/she finds that Violas work in this project workgroup covers multiple research areas.

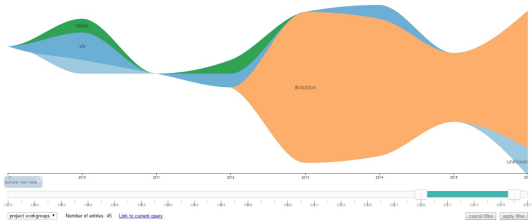
As shown in the example various attributes enable the exploration of questions like keyword co-occurrence, collaboration and driving themes, all in relation to time. Without the visualization most of the insights found in this exploration task would be hidden in the result list. As the list shows individual publications, one at a time. This visualization therefore provides a complement to the existing simple text box search, which can provide a targeted search function, but gives very little support for overview, exploration and analysis of the database.



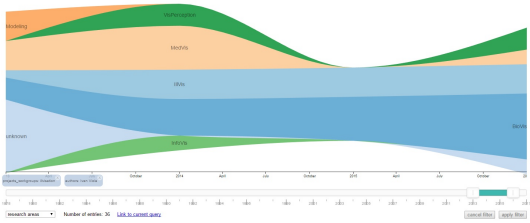
(a) Project workgroups



(b) Authors of the IllVis project



(c) Projects Ivans publications are associated with



(d) Research covered by Violas publications in the IllVis project area

Figure 1.1: Introductive exploration example.

CHAPTER 2

Related Work

Stream I/O, the user interface of this thesis, uses time-dependent visualization alongside a faceted filtering approach to manipulate corpus data. By this definition the application falls into three research areas, therefore the chapter is divided into three sections, each providing an overview of existing algorithms and methods. First, the Streamgraph [BW08] and related visual encodings are presented. The second section comprises corpus visualization tools which use topic mining together with metadata of corpora. Finally, related methods are investigated that use faceted browsing approaches.

2.1 Streamgraph and Related Visualizations

The chosen visualization for the user interface is the Streamgraph by Byron et al. [BW08]. This visualization builds onto the principles of the layer area graph developed by Harris [Har00] (see Figure 2.1). Layer area graphs are two dimensional visualizations, where the x-axis encodes time and the y-axis a quantitative data attribute. Area charts depict a time series relationship, however, unlike line charts, they also visually represent volume. This volume is constructed by connecting single data values of a particular variable over the time. A layered area graph is a stacked visualization where the volumes, called layers, are drawn upon each other. All subsequent layers are drawn relative to the layers below, therefore the order of layers influences the visual appearance of the individual layers drastically. The advantage of using a layer area graph is the fact that they emphasize the total sum of values while providing information about the parts that constitute it.

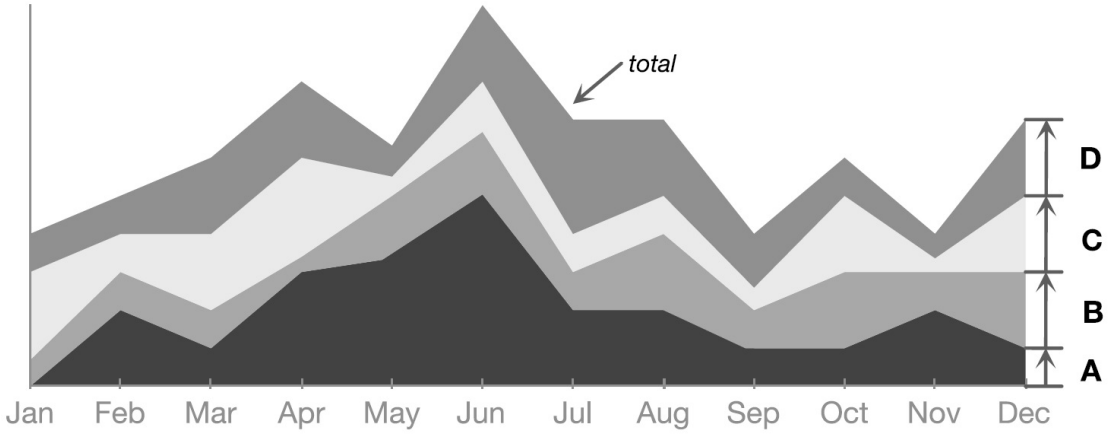


Figure 2.1: The layer area graph from Harris et al. [Har00] gives an visual representation for the total sum of values with its straight bottom baseline.

Based on this visualization, the ThemeRiver technique was developed by Havre et al. [HHN00]. The layers of the ThemeRiver are computed in a similar way to the layers of the layer area graph. However, in the ThemeRiver, the layers are interpolated to simplify the exploration of the layers over time. Also rather than drawing a layer on the bottom of the visualization and stacking the subsequent layers on top of it, the layers of the Themeriver start in the center of the x-axis. Following layers are added on top or below the first current. Big gradients, which are the result of the classic stacking method, are halved by this stacking method, because in contrast to the stacked bands, the currents width can expand in both directions of the y-axis. The area outlined with black in Figure 2.2 shows this phenomenon for an ascending and descending gradient, the center and bottom graph show the comparison mentioned.

The Streamgraph from Byron et al. [BW08] is a generalization of ThemeRiver, which was introduced explicitly for changing term frequencies over time. In principle, the construction of a Streamgraph is exactly the same to a ThemeRiver, although some aspects are refined to improve the legibility and aesthetics of the resulting graph.

In their research, Byron et al. found that the overall geometry of the graph is determined by the shape of the baseline, which is the bottom of the lowest layer, and the order of layers. Using a layout that is symmetric around the x-axis, which means the bottom of th lowest layer has the same form as the top of highest layer, was suggested by Havre et al. Aside from aesthetic qualities of a ThemeRiver, the proposed layout has positive effects on legibility by minimizing important quantities. In particular, each point on the graph silhouette is as close to the centerline as possible and the gradient of the silhouette is as small as possible. These two quantities can be explained by the fixed thickness of the graph. As the graph expands equally in both y directions the thickness is halved between the two directions. Differences in thickness are also split between those two symmetrical sides which causes the gradient being as small as possible.

From a mathematical point of view, these two quantities are minimized in a sense of total sums of squares. Byron et al. took this concept and applied the same optimization criteria to each layer overall. They defined a formula that minimizes the distance from the centerline and the slopes, weighted by thickness, for all layer edges. With this formula the readability of the individual layers is increased and the wiggle in the layers is decreased significantly, which can be seen in Figure 2.2 by comparing the first and second graph. As the thickness of the layer determines the weight of individual layers, the minimization of those properties favors thicker layers which are visually more important.

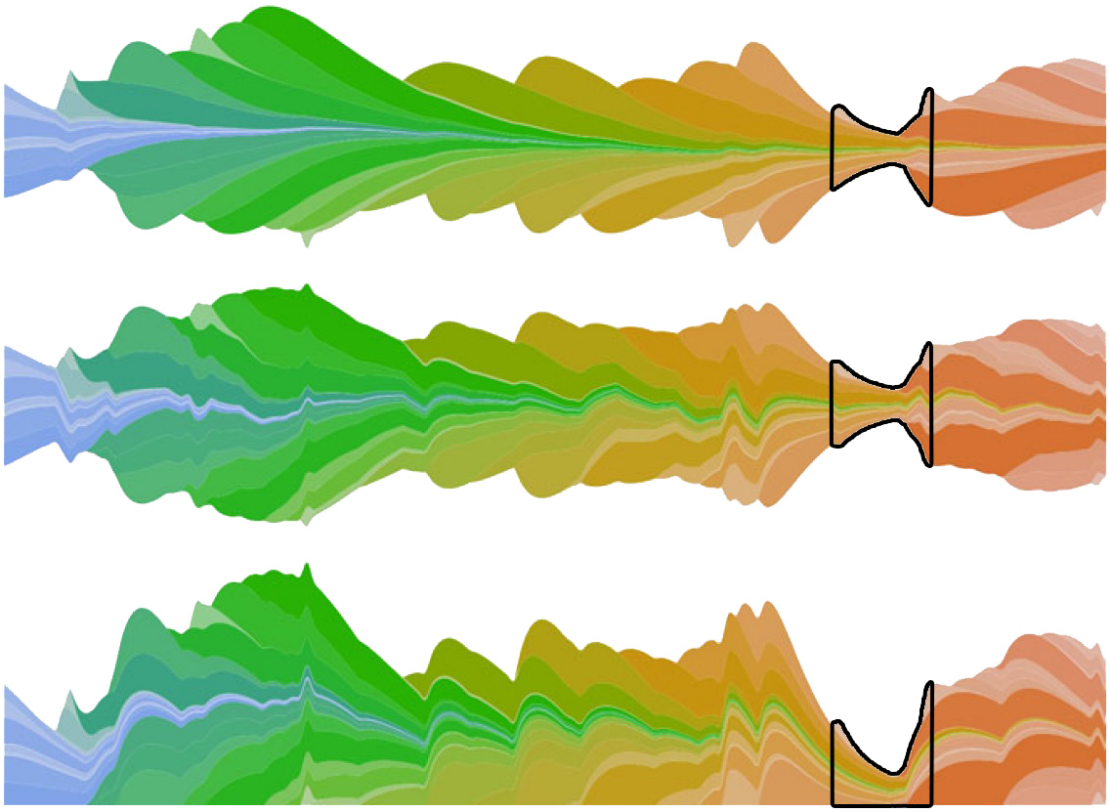


Figure 2.2: This comparison from Aigner et al. [AMST11] survey shows the effects of the different layout algorithms compared: Streamgraph design (top), ThemeRiver layout (center) and traditional stacking (bottom).

Besides changing the geometry of the layers, Byron et al. also proposed a way of order layering that deals with some problems, a ThemeRiver might have. If new layers are always added on top, a downward diagonal stripe pattern in addition to an upward angle to the overall silhouette is introduced into the visualization. This can be seen in Figure 2.3 . To prevent this, layers are ordered "inside-out", in which early layers are placed in the middle, with later series added at the top and the bottom. This approach wields three benefits, besides the removal of the stripe effect. First, it places the new layer bursts at the outside of the graph, where they disrupt the layout of the other layers the

least. Second, the silhouette of the graph is the most prominent area, because there is more contrast, so this approach guides the awareness to new layers in a given time period. And finally this procedure prevents the layout from drifting away on the x-axis.

Figure 2.3: The downward diagonal stripe pattern described by Byron et al.[BW08]

.

In summary the Streamgraph is a layered area graph visualization, where important aspects for legibility and aesthetics are improved. This includes a changed algorithm for the geometry of the specific layers of the graph as well as the order layering. The result is an improved time-dependent graph that can show the quantities of individual attributes with a time dependency while also conveying their sum. This property along with its appreciated aesthetics and legibility [BW08] makes the Streamgraph a solid choice for the application.

2.2 Time-dependent corpus visualization techniques

In the field of interactive visualization, our work is related to time-based, corpus meta data visualizations, which help users to get an overview of the documents. Researchers have developed a plethora of tools that use metadata for the visualization. Metadata in general is data which holds information about a certain set of data. In the case of this thesis, the visualized database attributes hold information about the publications written at the institute, thus they are metadata of this corpus.

TIARA by Liu et al. [LZP+12] is a visual text analysis tool that also uses metadata for the visualization. Their tool includes new text summarization techniques as well as a new visual design to sum up a large text corpus. They use a LDA-based (Latent Dirichlet Allocation) topic analysis technique deriving a set of topics to summarize a collection of documents and their content evolution over time. In the process of creating keywords and ranking them, metadata can be very helpful. For instance, Liu et al. used the metadata of emails. Metadata whether an email has been read or replied to may imply importance. They incorporated this additional domain information in their ranking model.

The resulting topics can be explored in two linked views and a collection of interaction tools. To make the topic output easier to understand, it is visualized in the topic view by an augmented Streamgraph, where the layers of the graph represent the derived topics. These topics are described by a keyword cloud, where the keyword cloud is representing the content of the topic as well as the content evolution over time as seen in Figure 2.4. The topic view enables interaction through adding topics and showing additional keywords, if they are needed to recognize a topic. The document view additionally shows snippets that contain the selected topic keyword. As the views are linked, it allows the users to analyze documents both top-down (from topics to individual documents) and bottom-up (from individual documents to topics).

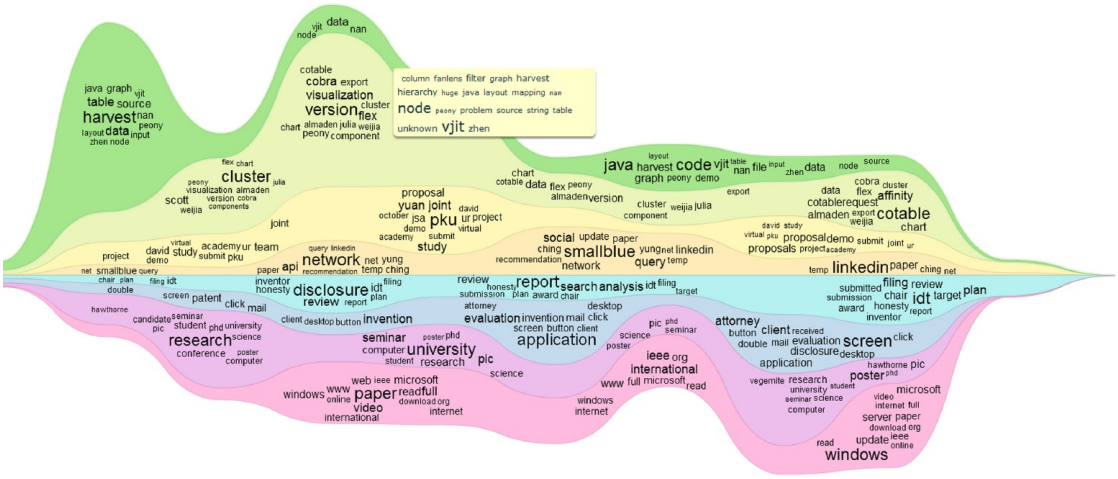


Figure 2.4: Augmented Streamgraph visual design from Liu et al. [LZP+12] which shows keyword clouds as labels.

It has to be mentioned that TIARA generates topics from their corpus and displays them instead of displaying metadata, making it very different to Stream I/O, the application of this thesis. Despite this difference, it is interesting to compare because both applications show an overview on the given corpus over a time period and can be used to solve the given problem statement. TIARA focuses on the summarization of text corpora into topics. For tasks that revolve around finding and exploring topics, applications like TIARA, that use topic mining, are superior. Stream I/O, on the contrary, represents the database through multiple views of metadata. On topic mining tasks the keywords of authors might not give as much insight, however further coupled views can provide information on how different metadata attributes relate to one another. In summary TIARA uses the augmented Streamgraph to show one hierarchical facet, while our application uses the Streamgraph to visualize and navigate along metadata dimensions describing the underlying publications. TIARA excels the capabilities Stream I/O on topic tasks, in contrast Stream I/O tasks can be more general as the metadata provides information besides the topic.

Another different approach of visualizing keywords and their evolution over time is TextFlow from Cui et al. [CLT+11]. TextFlow consists of two main components, namely the topic mining and the visual representation to present their results. Extracting topics from a corpus is done with Gibbs sampling procedure based on HDP (Hierarchical Dirichlet Process). Along with the topics, three different features are generated as well. The focus of this work is the first feature, the evolution of topics, the researchers introduced gaps into the simple stacked graph layout to emphasize the merging and splitting. Thus the layers of the topic flow merge and split into several branches according to the merging and splitting of topics. TextFlow's visualization is further augmented with glyphs that signalize the second feature critical events like the source and the sink of a topic as well as splitting and merging events. For the last feature, keyword correlation, keywords are represented as polylines. The distance between polylines encodes keyword correlation, when the polylines weave together the keywords co-occur.

The resulting visualization can be quite complex when all these visual cues have to be taken in. Cui et al. developed an interactive exploration approach which reduces the visual complexity by introducing an exploration pipeline. This pipeline allows users to interact with the graph at certain points (seen in Figure 2.5) through selection and exploration. Progressively various features are added, the user can filter at each step. All selected elements are further explained by two linked views providing additional information to aid the filtering process. When this process is done, the complete visual representation will be displayed and users can explore it in its entirety.

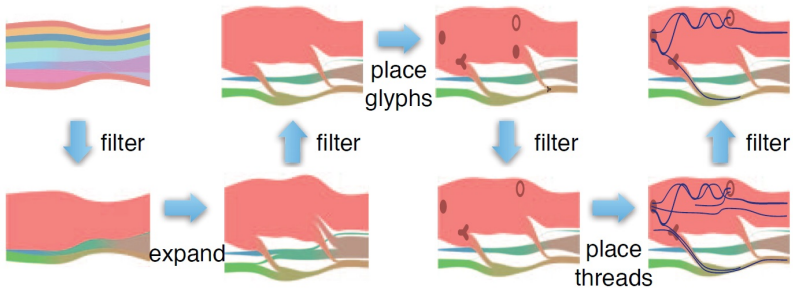


Figure 2.5: Cui et al. [CLT+11] progressive exploration pipeline: The topic flow is generated first then the three topic features follow, filter arrows indicate interaction points in the pipeline.

TextFlow focuses on evolution of topics over time and uses topic mining to enable deeper investigation of keyword correlation. In comparison to our Stream I/O application, which gives an overview from multiple views, TextFlow has more specific topic information at hand. To convey this information the visual encoding is more complex in comparison to the Streamgraph used in our application. Thus the researchers had to compromise on the goal to convey the sum of topics, as they introduced gaps between layers to show merging and splitting. Also the polylines conveying the keyword correlation are a trade-off, limiting the number of displayable keywords before the graph gets to cluttered.

Finally a closer look is given onto KeyVis by Isenberg et al. [IIS+14], which focuses on publication keyword analysis. The researchers collected three data sets of keywords and analyzed them to find major research themes and observe the evolution of keywords over time. To analyze these keyword data sets, they filtered each data set first. This manual cleaning pass consolidates keywords which were the same such as singulars/plurals or acronyms. After this, a manual expert coding of the keywords was done on one of the data sets in order to find higher-level clusters of keyword topics. This procedure then wielded the final keywords that were used in the analysis. There the researchers calculated clusters from the expert coding, encoded them in keyword-occurrence matrices and performed hierarchical clustering. With the finished clustering they generated a network graph to show the cooccurrence of the keywords. Then a webpage was created to enable the browsing of the created expert keywords and related papers.

In contrast to the Stream I/O tool presented in this thesis, the KeyVis application revolves around the analysis the researchers have done on the data set. The data sets used in the application are either based on established keyword taxonomies (e.g: IEEE, INSPEC terms) or were encoded by experts of the field. In comparison, Stream I/O does not include manual filtering. The employment of experts to do the encoding as well as the needed time frame exceed the scope of this thesis. Furthermore the KeyVis application is focused on cooccurrence rather than the evolution of the metadata over time, a visualization completely different from the Streamgraph was chosen. On the KeyVis webpage the researchers incorporates a histogram, therefore the evolution over time aspect is visualized by an approach similar to the Streamgraph.

2.3 Faceted browsing

Exploration in the Stream I/O application is realized in terms of faceted browsing. It allows looking at the same data from different conceptual dimensions and enables an incremental refinement of structured data by filtering. Faceted browsing is a concept that is widely applied to a variety of fields such as e-commerce sites (e.g., Amazon [Ama], Ebay [Eba]), bibliographic databases (e.g., IEEE/IET Electronic Library [IEE]) and multimedia libraries (e.g., Google Images [Goo]).

An approach from the field of image retrieval that is similar to this thesis is the Flamenco interface by Yee et al. [YSLH03]. The researchers present an interface that lets the users explore an image database by navigating along conceptual dimensions which describe the images. During the exploration, the user progresses through three stages, which define the interfaces tools, each designed to aid the user with the task at hand. At the starting stage, a search box and metadata terms, which can be used to filter, are used to define an initial query. Then an initial result is gathered and the next stage is entered. There metadata terms are pushed to the side to show the current result sorted by facet, when the user selects an image the interface progresses to the final stage. In the end stage the image is displayed along with a hybrid tree that shows all metadata terms assinged with the item as links. They can be used to expand the query in further directions.

In contrast to the application of this thesis, the researchers use a searchbox and metadata terms alongside with a count indicator sorted after facets to explore the data. In comparison to the exploration through the Streamgraph, this interface has the advantage of showing multiple facets simultaneously. However, the visual representation of the occurence count is interpreted easier in the Streamgraph as it is encoded visually and the evolution over time is shown additionally.

Due to the effective approach to faceted browsing, it also found its use as an interaction concept in corpus visualizations where metadata is available. Both tools introduced in Section 2.2 got interactive filtering methods. The TIARA framework applies the faceted browsing approach to filtering. The tool contains multiple facets, derived topics form a hierarchical facet where different topics are on a higher hierarchy as the individual keywords that represent the topic. Further facets are the different metadata attributes which can be used for filtering. All facets are represented in a juxtaposition with a designated view or menu in the tool.

The TextFlow application introduces a faceted filtering approach. Topics and keywords form a hierarchical facet while the three TextFlow features, described in Section 2.2, are further facets. In contrast to the TIARA tool, the facets are displayed in a superposition in the same view, as they are represented with their own visual elements. The facets are filtered in a progressive pipeline as shown in Figure 2.5. This has the effect, that later filtered facets are more restricted due to the previous facet filters. Because not all keywords are represented in the topic flow visualization, the researchers added two views that are linked to the selected topic. The word cloud view and timeline view, which shows meaningful text snippets, allow filtering of keywords that are not represented in the topic flow.

In comparison, Stream I/O visualization presented in this thesis consists of five facets that are defined through the different metadata that can be visualized. The filtering query is produced sequentially by filtering the database with one facet at a time. As the filtering process is sequential, it is made up of a conjunction of different facets, the filtering components. These components are saved and displayed separately to the user to display and record the sequence of actions the user has done within the session. It was chosen to display the single facets separately as it allows the elimination of single facets from the query. The partitioned filter approach enables an easy way to return to the previous query while also offering an intuitive way to manipulate the different facets individually.

CHAPTER 3

Stream I/O

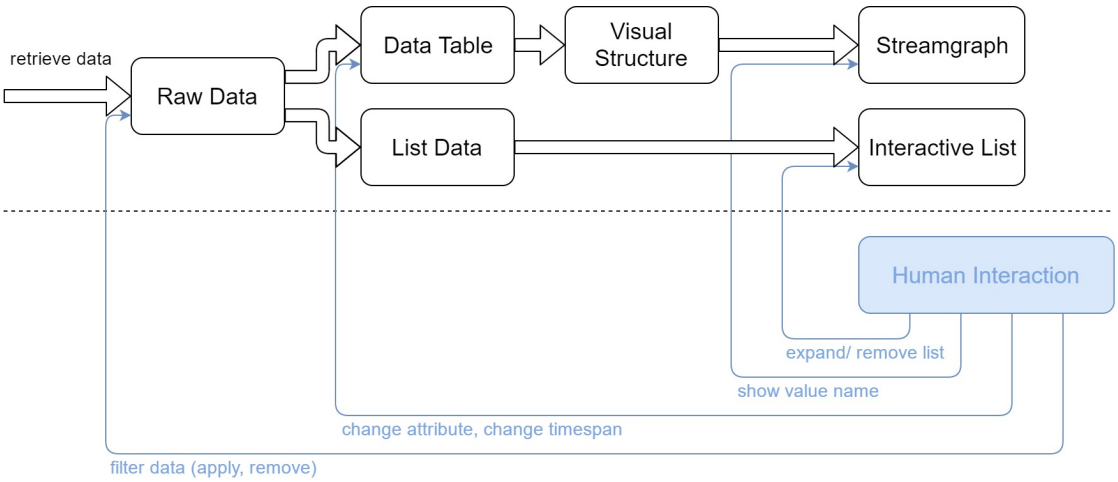


Figure 3.1: Basic pipeline of the Stream I/O application (user interaction in blue, pipeline in black).

The Stream I/O user interface works by following basic steps, shown in Figure 3.1. At the start of the application, the data is retrieved from multiple JSON files, and some preprocessing steps are done, like null-value handling and normalization of date, in order to make the retrieved data consistent.

From this data, the data tables for the Streamgraph as well as the interactive list are generated. This generation process transforms the data, the single data entities are nested in a structure that sorts them after specific data attributes. In case of the list data table, the data is sorted after year of publication, so that the algorithm computing the list can rebuild this structure in html as expanable containers. Following the creation of the Streamgraph, these data tables are of an entirely different structure as single data

entities are transformed into another visual representation. In the graph the publications are represented as summations, this table hierarchy therefore groups them based on attribute and year. From this step onwards, only the occurrence count of the attribute values is saved.

These number of occurences are subsequently used to build stacks for the Streamgraph, which are then interpolated and specially arranged to make the layer easier to be understood. At this stage, the Streamgraph is displayed along with the interactive list and further interaction tools, which marks the end of the pipeline.

Through interaction, the pipeline is set back to specific stages based on the complexity of the manipulation. The application manipulates the data mirroring the interaction of the user, steps into a specific stage and works its way up the pipeline by following the steps that were briefly mentioned before.

Further specification for the used algorithms and libraries are given in Chapter 4. The following section will focus on a more specific description of the pipeline while explaining design choices that were made during development.

3.1 Raw data and preprocessing

To create the visualization, the data has to be retrieved first. In case of this database, the data is available as multiple JSON files. One file contains the publications and another contains additional information to retrieve the authors. The JSON file contains therefore an disordered set publication entities, each consisting of up to 23 attributes, which need preprocessing to be consistent. Depending on the age of the entry, filled out information and type of the publications, the number of attributes may vary.

An example of such a publication entity can be seen in Figure 3.2. In the preprocessing, the string of the publication date is normalized, as there are multiple attributes for the publication date and the entries have no consistent syntax, because they are manually entered without proper restrictions on the input field. After this step, the JSON file with the author data is retrieved and joined with the main table.

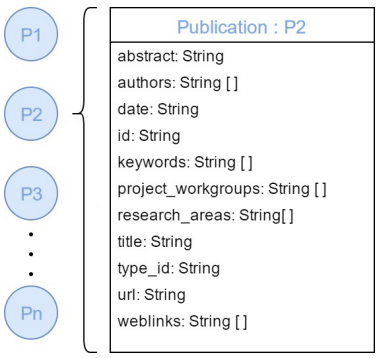


Figure 3.2: Example of a publicaion entity which shows possible attributes assigned to it.

3.2 Reorganizing Data

When the preprocessing is done, the data is reformed to reflect the visual mapping. In this specific case the Streamgraph visalization with its mapping was chosen because it shows the frequency of publication meta data over time. Thus providing a good overview of the publication activity of the institute in whole. At the same time the interaction tools provide filter options to focus on special people, areas or publication types in particular.

Shortly characterized, the publication data consists of abstract data that has no spatial aspect but has a temporal aspect, that explains the decision for a time oriented visualization. As there is no inherent spatial information or a distance function for the concrete values of an attribute, the temporal attribute is used to generate a quantitative model and a suitable mapping of data to the screen.

To summarize the characterization of the time domain, the data is mapped to a discrete, point based, linear time domain. The date of publication is treated as an instant time primitive. There is one chosen granularity, namely the year, which results in an determinate specification of time, as for the year the complete knowledge of all temporal aspects are present. Both characterizations show that the Streamgraph fits the data, because it is a visualization that emphasizes the time aspect in abstract data.

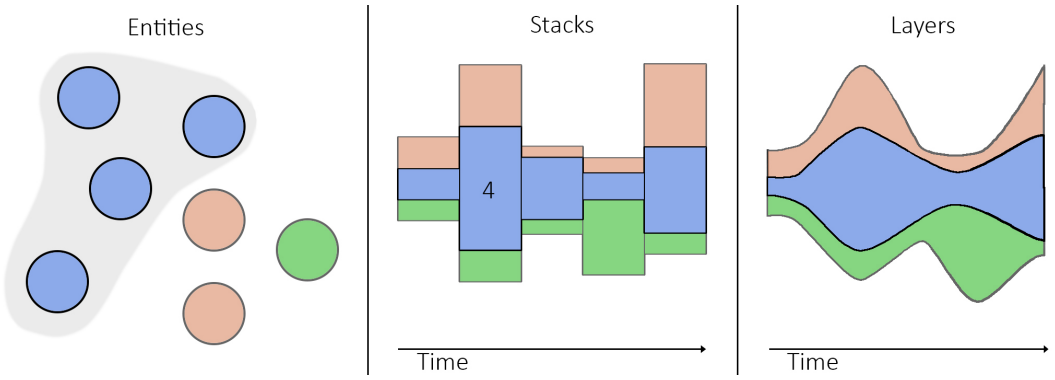


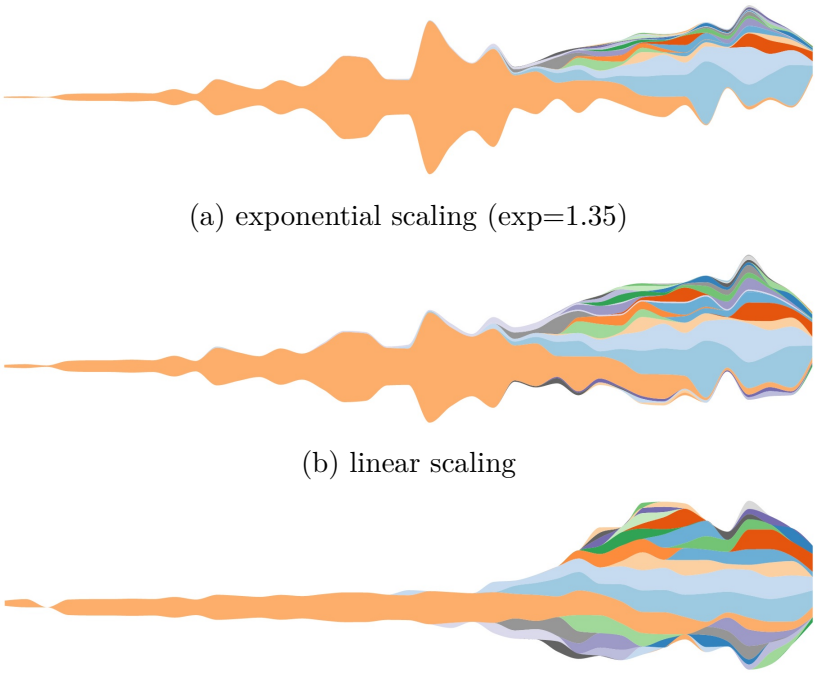
Figure 3.3: Evolution of the publication data from singular entities to a Streamgraph.

The visual encoding of the Streamgraph changes the single publication entities into layers that represent a certain attribute over the time axis. To visualize the evolution of an individual value, the occurrence of this value is summed up for each individual discrete point of the time space. This results in individual vertical glyphs that emphasize the attribute value behavior at a certain point in time. The Streamgraph idiom shows derived geometry that emphasizes the continuity of the horizontal layers. Therefore vertical glyphs are transformed by a global computation, which enhances legibility of individual layers. In Figure 3.3 the whole process of the visual encoding is depicted. This transformation of the data reflects the Streamgraph encoding, a multidimensional data table is built ordered after attribute and year which sums up the occurrence of an attribute value at the time primitive.

3.3 Visual structure

From the described data table, the visual structure is derived to represent the individual vertical glyphs or stacks. During the construction of the visual structure, an energy function decides the order of the attribute value stacks onto each other and the shift of the baseline to minimize the weighted wiggle of layers. Byron and Wattenberg [BW08] stated that a big factor, which influences legibility, is the wiggle of the layers, because the layers beneath influence the form of the silhouette. Also their order strategy wields benefits. First, introducing new thin layers on the silhouette of the graph, the layout disruption is minimized. Second, the new layers are placed on the silhouette, which provides more contrast and guides the awareness to them. And third, this procedure prevents the drift of the layout away from the x-axis.

However, the graph geometry is just one of multiple properties defined by the data structure. When it is defined, a closer look can be made on the scale (see Figure 3.4 for comparison) used to display the stacks. For this application, a linear scale was chosen, as this scale preserves proportional differences. Exponential scales can emphasize the most prevalent layers and logarithmic scales can normalize the thickness of the layers. Because the application should emphasize the relations between attributes, these scales would not be the correct choice. Additionally we decided to normalize the height of the layers, as the combined height of the layers should always depict the sum of publications.



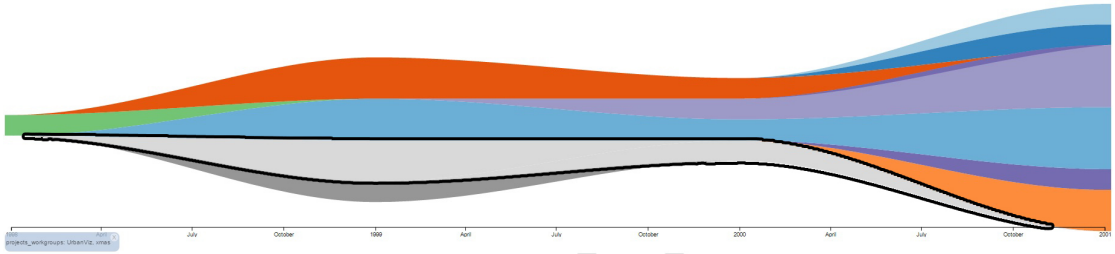
(a) exponential scaling (exp=1.35)

(b) linear scaling

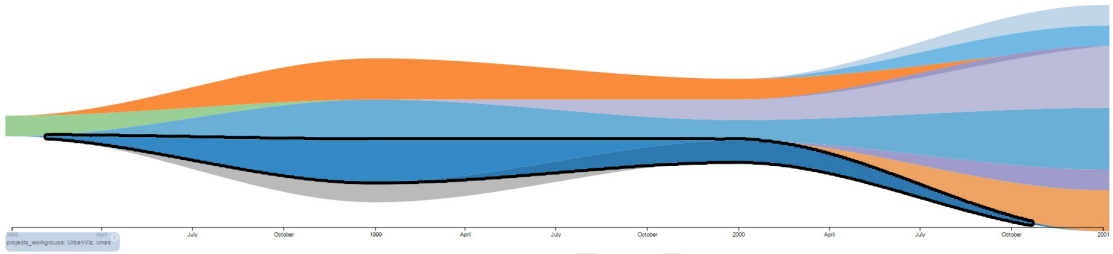
(c) logarithmic scaling (base=10)

Figure 3.4: Comparison between different y-axis scales

A further property defined by the visual structure are the color scale used for the coloring of the layers. As the values in an attribute are categorical, a scale is used that reflects this point. More precisely, a scale which includes color specifications and designs developed by Brewer [Bre]. This scale offers twenty different colors, which is insufficient in this use case as attributes in the database can take on up to 1600 different values. When the Streamgraph is filtered it can happen that neighboring layers have the same color, as in the example in Figure 3.5, which can result in errors in the analysis. To get rid of this problem, the color scale is shifted slightly into brighter and darker colors in an alternating fashion.



(a) Brewers [Bre] 20 categorical color scale



(b) Augmented categorical color scale

Figure 3.5: Comparison between different color scales.

At last, to derive the visual structure of the final Streamgraph, the individual stacks have to be interpolated to form layers. The individual start and end points of the stacks define control points for the topline and baseline curve that enclose the area of a layer. How to transform the discrete representation into a continuous shape depends on the curve algorithm chosen. For this application, a specific cubic spline, proposed by Steffen [Ste90], was chosen, because it preserves monotonicity in y, assuming monotonicity in x. The motivation behind this choice is that the cubic spline goes through the control points, thus representing the discrete data given precisely, while preserving the monotonicity of the data set being interpolated. The preservation of monotonicity has the effect that overlaps between layers are less common and smaller.

3.4 Interaction concept

The previous sections discussed the generation steps of the visual representation and which aspects had to be considered. This covers the computational side of the pipeline, now a closer look is taken onto the interaction that enables an analysis of the data. With respect to the users, it is not exactly known whether the generated visual representation is effective with regard to the task the user is trying to accomplish, thus interaction tools are provided. The interaction process helps users to understand the applied visual mapping through realizing the effect, a parameter change has, and filtering out hidden patterns.

Also the data set provided is big and complex and will constantly increase in size and complexity through time. Users are only able to comprehend a fraction of the available data at a time, they need tools to reveal the data which is important for their use case. The big problem of visualizing the database is split into smaller facets that allow to focus on relevant data aspects and particular tasks individually.

For a basic coverage of use cases the following interaction tools where chosen, the Streamgraph can be manipulated on the time span, the shown attributes and the shown attribute values. The manipulation of attribute values, the individual layers of the Streamgraph, is implemented through a filtering. The dynamic list under the Streamgraph is linked to the visualization. If detailed information is needed, it can be expanded or reduced.

All interaction tools trigger different interaction loops, as it is dependent on the interaction if the needed information is already available or has to be computed. Interactions as hovering a attribute value name, changing the time span and expanding or reducing the list is based on already available data resulting in small loops in the pipeline. Changes in attribute lead to bigger loops, as only the displayed attribute is computed in order to avoid wasting computational power. Previous computations of attributes on the same data are saved resulting in more economical interaction loops, because the same layer can be reused. When filtering is done, the pipeline returns to the raw data and builds the Streamgraph table and dynamic list table from scratch, resulting in an almost complete rerun of the pipeline. This procedure was chosen, because the filtering of the data can be done once and then the already implemented code can be reused to generate the Streamgraph again.

The addition of the interaction loops conclude the pipeline of the application, now more insight is given on the exact implementation of the interaction tools. Figure 3.6 shows the basic structure of the user interface.

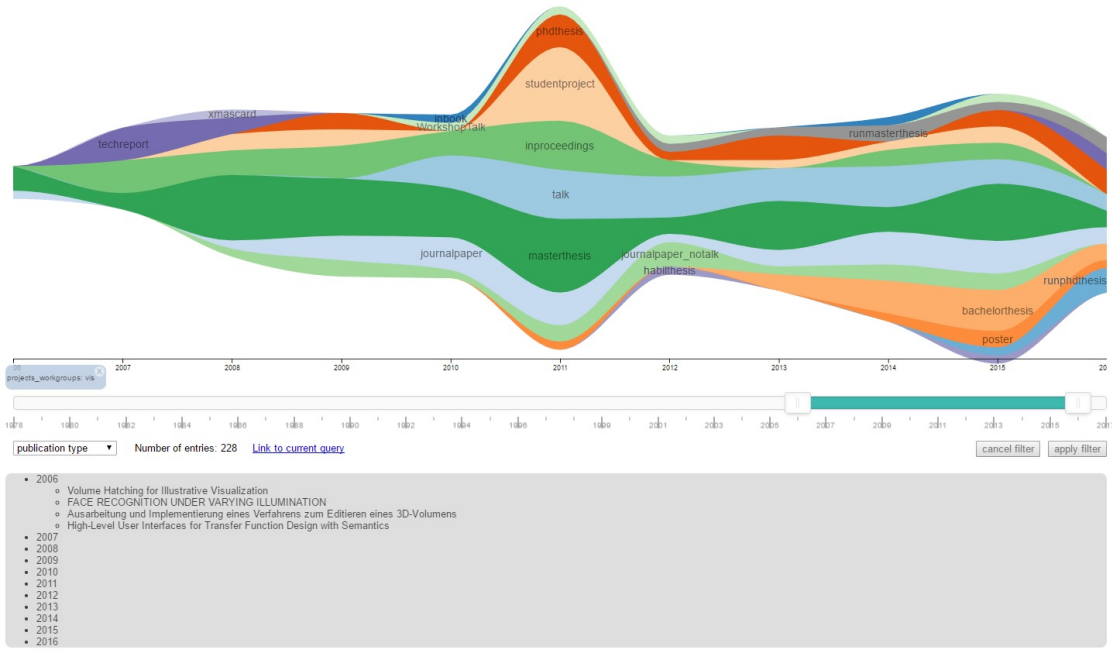


Figure 3.6: The user interface of the Stream I/O application.

The Streamgraph is directly labeled at the thickest point of each layer when a layer reaches a certain thickness. To show further labels the user can hover over the layer, then the attribute value of the layer is displayed on the left upper corner. Limiting direct labeling to a certain layer thickness has the purpose to minimize label occlusion. Showing labels on hover is a solid choice for thin layers as the attribute value is displayed clearly without occluding the silhouette of the layer.

The interaction tool for the manipulation of the time span and a dynamic list are implemented with the "overview, then detail" design philosophy from Shneiderman [Shn96] in mind. The manipulation of the shown time span is realized through a range slider with two handles, the positions of the handles on the discrete bar specify the visualized time range. As time flows in this encoding from left to right, the left handle corresponds to the start of the time span and the right to the end. When they are used, a transition animation is shown from the old state to the new one. This helps the user to keep the relation between the two states. Finally the detail-on-demand is provided through the interaction with the dynamic list under the visualization. This list is updated every time the user manipulates the data through the other tools. So it provides the detail to the shown layers, to show the user what their query found.

The Streamgraph can only display an attribute at a time with the single layers representing the attribute values. For many questions the interplay of multiple attributes is interesting, thus the Stream I/O adds an interaction method to change the attribute that is displayed to a selection of possible attributes. For this task a drop down menu can be found in the user interface displaying all available attributes. When another attribute is picked from the list, the user interface is refreshed to show the Streamgraph of the new attribute. In contrary to the time span manipulation, it was decided to change the view without animation, because this would imply a non-existing link between layers of the same color of different attributes.

The enabled attribute switching is not sufficient to enable analysis of relations between different attributes. This depends on the fact that the relation between the whole set of layers is shown in the Streamgraph, the relations between the single layers from different views are still hidden. An interaction tool was needed to satisfy this need, thus filtering, a concept very close to brushing and linking, was introduced. Instead of marking layers that are selected in multiple views, filtering removes data that was not selected in the filter selection process. As an example, a user could be interested in a certain authors publication history. This can be done through selecting the authors layer in the author view and then filter with this selection. Then a new Streamgraph is drawn with only the data that was specified by the user defined filter (e.g., all publication that were published by the author).

Such an approach results in many advantages, the number of layers is decreasing, resulting in less clutter and more space that is used to visualize the information the user is interested in. When the concept of hiding unnecessary data is chosen, the resulting graph silhouette shows patterns that are relevant for the task at hand, as all layers are drawn on top of each other. Resulting in a view where the relation of the filtered layers can be analyzed by the slopes of the layers, the trend of the whole filter can be seen through time. And the removal of layers has positive effects on the computational load. It should be mentioned that this approach does not provide focus + context.

Finally, to enhance the selection of layers for the filtering process a multi selection tool was implemented. Clicking into the SVG and dragging the mouse creates a selection frame. Layers in the frame are selected when the mouse is released. Some attributes create over one thousand layers when they are unfiltered. So, the multi selection tool simplifies the selection of many layers immensely.

CHAPTER 4

Implementation

Stream I/O is programmed by using the programming language Javascript. The D3 library [Bos] is a central part of this application, as the application has an emphasis on the visualization.

4.1 Libraries

To create the functionality of the application as described in Chapter 3, additional libraries are used to support the implementation process.

D3 [Bos] is a library that provides the tools to enable the creation of an interactive visualization. It provides the functionality to import data from any common file format, bind the data to their Document Object Model (DOM), apply data-driven transformations and then visualize the results. The majority of functionality is implemented using D3, in particular this includes the data acquisition, the data manipulation, the generation of the visualization, the animation transitions and the events triggered trough interaction.

NoUiSlider [noU] offers a lightweight JavaScript range slider, which is used for the time span interaction. The library includes many functions to customize sliders, in particular the number of handles, the easy reading and setting of values and the tapping and dragging interaction with the slider bar which are used.

D3.EZ [D3.] is used for the functionality of the expandable HTML list. Together with D3, the HTML UL/LI list is generated, which is then used to display the currently queried publications in an ordered way.

Require [Req] serves as a JavaScript file and module loader, which is optimized for in-browser use. Primarily this library was added to the application to improve the quality and readability of the code by partitioning.

2D [Lin] adds to the SVG functionality of JavaScript to enable intersection calculation between various shape elements. This library is used to calculate the intersections between the multi selection frame and the layers of the Streamgraph.

4.2 Working Pipeline

The following section describes how the previously introduced steps in the working pipeline are implemented by using JavaScript and the mentioned Libraries. For the sake of simplicity and consistency, the naming convention of the previous chapter is maintained.

4.2.1 Raw data and preprocessing

Retrieval of the necessary data stands at the start of the application. For this task two JSON files are utilized containing publication and author data separately, both were downloaded from the institutes website. To download the publication data JSON file, the database can be queried with an empty query, the author data is at the time not publicly available. As a first step both JSON files are processed. This step is done asynchronously with the function `d3.json()` and a callback. Both Files contain the data in name/value pairs as usual for JSON Syntax. These pairs are translated into Javascript properties, Figure 3.2 shows how a publication entity is structured. Further details on the data format can be found in Section 3.1 . When both files are available, they are joined. Before the data is reorganized into a data table that reflects the visualization, inconsistent data has to be filtered out. In the case of this application the only missing data the visualization cannot handle is a missing publication date. Any placement on the x-axis would suggest that the publication date is known. To keep most publications in the visualization all attributes, that give information about the time span the publication fits in, are searched. The algorithm searches for valid substrings in these fields and prefers earlier dates, all publications that cannot be matched to a date are discarded.

4.2.2 Reorganizing data

When the filtering is done, the representation of the data changes from single publication entities to continuous layers that represent a certain attribute over time. In order to achieve this, first data tables have to be generated from which then stacks can be created. In a last step these stacks are interpolated to create the Streamgraph visualization. This process is realized by methods already existing in D3 and is visualized in Figure 3.3 . To generate the aggregation data tables for the stack creation, the publication entity data is itarated over. Each occurrence of an attribute value is added up in the aggregation table which is ordered after attribute and year. It has to be mentioned that this data table has to be a dense table, if there is no publication that fits the attribute value at a certain time the value is zero. The summation of the values is done that way because the `d3.stack()` (D3 4.0) implementation, which creates stacks, needs complete tables.

4.2.3 Visual structure

Creating stacks that use the the graph geometry suggested by Byron & Wattenberg [BW08] can be done through the addition of the `d3.stackorderInsideOut()` and `d3.stackOffsetWiggle()` functions. These two functions implement the suggested stack order and offset to produce the graph geometry as it is described in Section 2.1 . After the stack calculations finish the interpolation of the stack boundaries follow to derive the final Streamgraph. The individual top and bottom points of the stacks define control points for the layer geometry. As mentioned in Section 3.3, a variation on the cubic spline was chosen proposed by Steffen [Ste90]. To derive the layers from the stacks, the function `d3.area().curve(d3.curveMonotoneX)` is used as it implements the needed functionality for this task. Then the coloring is done to finish the computation of the visual structure. Every value of a certain attribute has a distinct color, which stays the same during the session by assigning a color at the first data retrieval and storing it. As the values of all displayed attributes are categorical, the color scale developed by Brewer [Bre] is used to reflect this observation. D3 has also included this scale in their library which can be found under the variable `d3.schemeCategory20c`.

4.2.4 Interaction concept

The visual structure concludes the computation of the visualization, now a closer look is taken onto the interaction pipeline and the components it is made of. As mentioned in Section 3.4, all the interaction tools trigger one of multiple interaction loops, where based on the interaction needed data is recomputed and then displayed. Depending on the pipeline stage (seen in Figure 3.1), the tools have to work properly. So a state pattern was adopted for the application that manages changes and ensures that the application runs correctly. There are three states implemented: display, filter and animation. Display is the main state of the application, in this state all interaction tools are available. Filter was implemented, because many of the `EventListeners` of the Scalable Vector Graphic (SVG) that display the Streamgraph are overwritten with new functions during the selections of attribute values to filter. Animation was added to block interaction while the SVG is changing. This stops the user from starting multiple interactions before the first one has ended.

The interaction tools managed by this state pattern are: the SVG, the publication list, the timeslider and the attribute drop down menu. Some insight into how these where implemented is given in the following paragraphs.

Filtering of attributes, labeling of layers and, most importantly, the visualization of the current queried data is done through SVG. Multiple `EventListeners` on the SVG implement the functionality that changes based on current application state. A `svg.on("mouseover")` listener shows the label of the layer and a `svg.on("click")` listener enables the selection of layers which initializes a state change to the filter state. Alternatively a `svg.on("mousedown")` listener initiates the multi selection tool, which uses line intersections with layers to determine which layers are in the selection frame.

Upon the use of the whole filter it is assembled from previously saved modular attribute filters.When the whole filter is created, the cleaned data of publication entities is filtered with `Array.prototype.filter()` function of JavaScript. For the publication data of the institute we got to the conclusion that attribute filters should produce the union of attribute values. This choice is motivated by the consideration, that filtering with an intersection on attribute level would yield results that are frequently almost immediately empty. This is especially the case with attributes that map one-to-one onto publications. However, the filtering between attributes was chosen to be an intersection of the attribute filters which enables to filter the data from multiple facets. The filter process generates for each attribute filter the union of publications with the selected values and then builds the intersection of these publication unions to retrieve the final filtered data set. With the filtered data set the whole pipeline is passed through to display the updated Streamgraph. The whole process from the user inputs up to the filtering pipeline is illustrated in Figure 4.1 . It can be noticed that in the pipeline filtered data is not saved to be reused. The motivation behind keeping the simple pipeline is that there is only one case where a filtered dataset can be reused. This case is setting an attribute filter onto an unfiltered attribute.

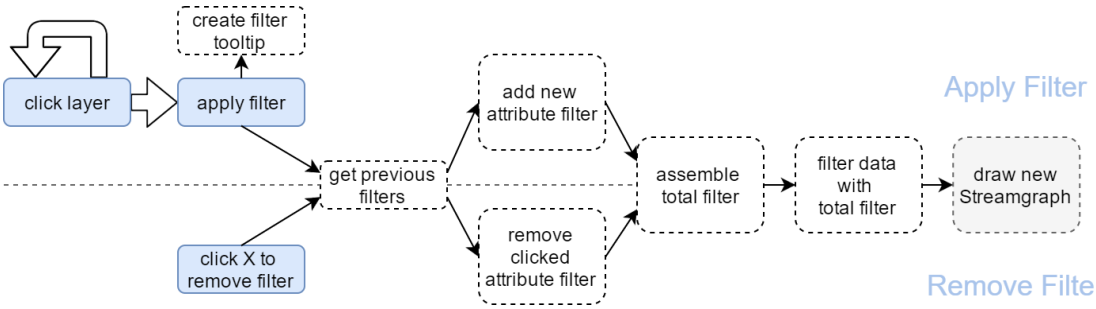


Figure 4.1: Blue boxes represent the user input, dotted boxes are the filtering pipeline.

Besides filtering, the second dimension of the Streamgraph can also be changed with a two handled slider. The NoUiSlider library provides a slider that is easily customizable. In the case of this application it was decided to use a two handled slider bar with discrete steps, the `start` and the `step` attribute of the noUiSlider constructor provide the needed functionality. The handles of the slider are connected by the use of the `connect` attribute, so the time range can be shifted with one click rather than moving both handles separately.

Under the application's visualization the publication list is displayed, which is a list generated through the D3.EZ library. The library gets the raw data filtered by the current query and generates a HTML list using the `<ul>` tag. All elements of the current data set dimension are added to the list with the `<li>` tag and get `EventListeners` to expand (or collapse) on click.

CHAPTER 5

Results

For the purpose of evaluation it was decided to analyze through concept walkthroughs to show how the visualization is able to solve various user tasks. To give a broad overview to possible user tasks, the formal task model from Andrienko and Andrienko [AA06] was used to determine a fitting task selection. On the first level, the Andrienko model is split into two classes of tasks, the elementary and synoptic tasks. Elementary tasks address individual data elements, whether they are individual values or individual groups of data. Synoptic tasks consider the data as a whole by representing the data in a configuration (i.e : a ordered sequence of attribute values). These tasks are divided into descriptive and connectional tasks. Descriptive tasks specify properties of a set of references or characteristics and a configuration aiming to describe the data as a whole by a pattern. The aim of connectional tasks is to go beyond the description and summarization of the data as a whole by explaining the driving forces behind the observed patterns.

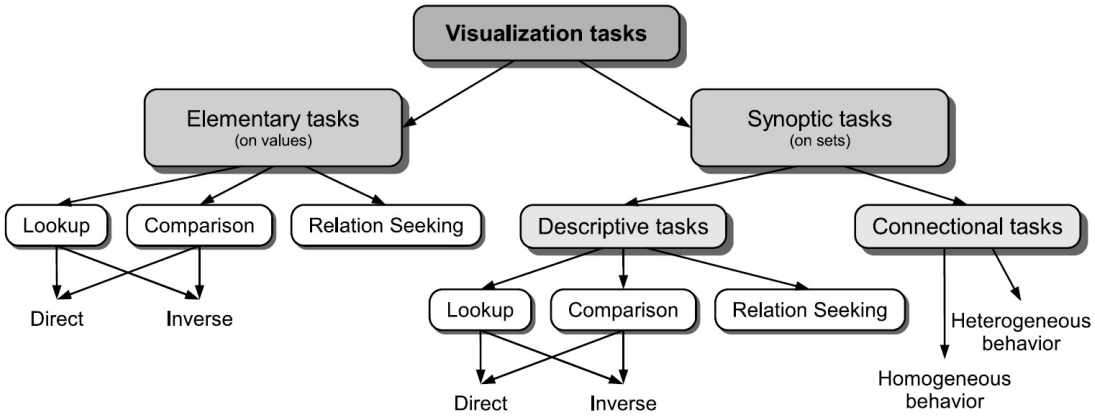


Figure 5.1: Formal task model from Andrienko and Andrienko [AA06] .

The qualitative results inspection method will be the walkthrough. It will comprise of an elementary task, a descriptive task and a connectional task. As it can be seen in Figure 5.1 , this selection gives an overview to possible user tasks defined by this model.

5.1 Walkthrough: Elementary Task

Elementary tasks involve references and/or characteristics that are dealt with as individual items. The question this walkthrough revolves around is the following: *Find all publications from the rendering group concerning with the rendering research area.* This question defines two constraints onto the publication data, one concerning the work group and the second limiting the research area. Setting a constraint as a filter involves, switching to the attribute with the attribute dropdown menu, clicking the layers to filter out (e.g: the rendering layer) and applying the filter with the "apply filter" button. Applying these interaction steps to both attributes results in Figure 5.2.

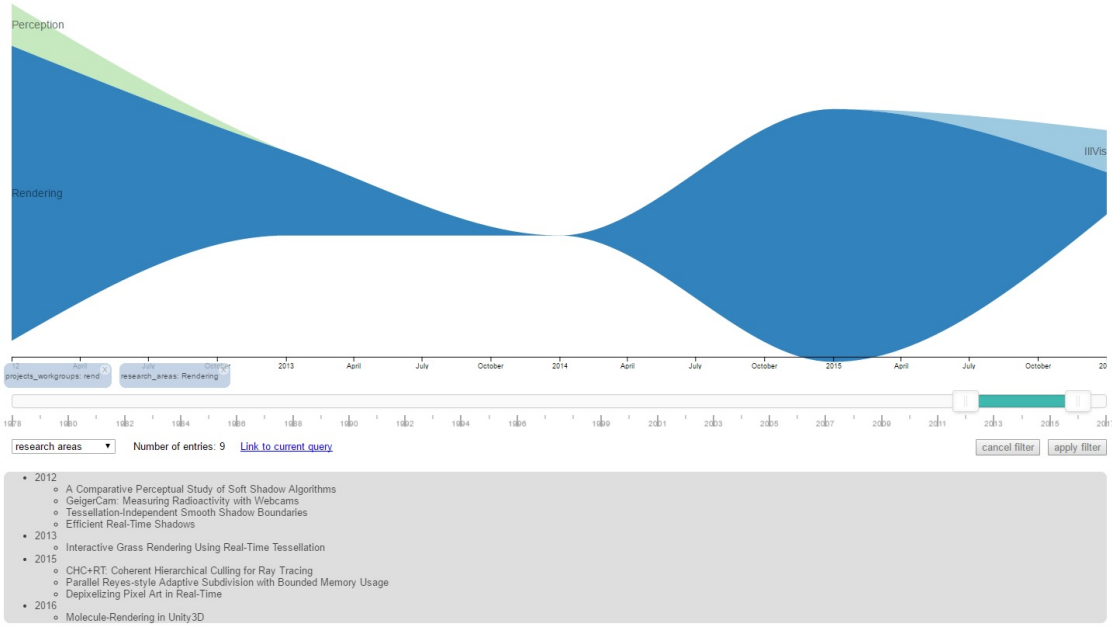


Figure 5.2: Publications from the rendering group in the rendering research area visualized and shown in a list.

This example shows that simple elementary tasks can be solved very effectively with the Stream I/O user interface, the list shows the individual elements of the result.

In case of this task, it was enough to recognize the task constraints and add them as filters which results in six mouse-clicks. Compared to the current interface which needs eight mouse-clicks and filling in the text-box for research area, the Stream I/O interface is marginally more effective. When the query is extended to involve multiple attribute values a much clearer picture emerges. For instance, one such question is: *Find all*

publications from the rendering group concerning with the rendering and/or perception research area. The question includes the intersection and union of multiple attribute values as both interfaces are designed to enable only one specific operation. Comparing both interfaces each new attribute value adds four clicks and a text-box for the current interface, in contrast to clicking an extra stream in the Stream I/O interface. So this example shows the Stream I/O interface scales much better.

5.2 Walkthrough: Descriptive Task

A descriptive task is defined through a specified property on a set of references. In this specific case we will compare two reference sets as a whole from multiple attribute views. For this task we are interested in students that finished their bachelor degree on this institute. *In particular, how the number of bachelor students, that are aiming for a higher degree on the same institute, compare to students that finish their studies with the degree.*

To visualize this data, first a filter on the publication type is set to show only bachelor theses. Therefore, the application view is switched to the publication type attribute and the bachelor thesis layer is clicked. With this selection the press of the apply button sets the filter. As the question revolves around the authors, we first select all layers of bachelor thesis authors (as in Figure 5.3) to then loosen the restriction onto the publication type. Selecting all layers can be done by clicking on the layers and applying it with the apply button. Alternatively, the multi selection tool can be initiated by holding down the mouse button. Dragging the the created frame over Streamgraph layers selects the layers for filtering. With this query, further publications are visualized, which the bachelor students have written.

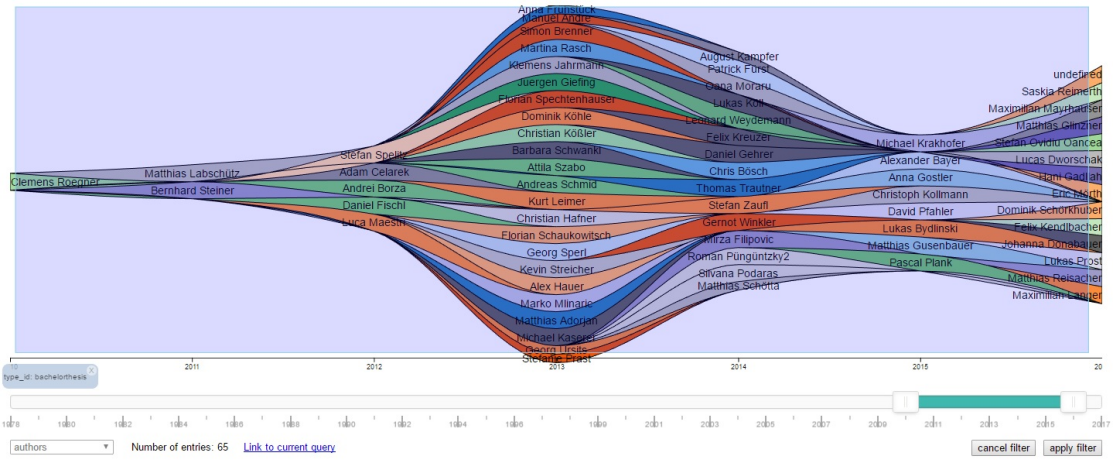


Figure 5.3: All authors that published a bachelor thesis are selected.

With the new filter the publication type attribute can be further examined. As the descriptive task revolves around academic degrees, we filter for all degrees and currently running theses for each degree. Using the same process as before the layers of interest are clicked followed by the apply button.

Through the visual encoding of the data, the Stream I/O application provides a configuration needed for the synoptic task. The configuration shown here is a sequence of attribute value occurrences over a period of time represented as a volume (i.e the layers). This volume now enables to answer questions concerning the behavior of data set as a whole over the entire time period or during subintervals.

As the task revolves around two sets, we will compare the form of the bachelor thesis layer with the form of combined layers of higher degrees. The result in Figure 5.4 shows that the students, who continue with higher degrees on the same institute where they had finished their bachelor degree, is increasing since 2014. Compared to them the bachelor theses were finished in the year 2013. Since then, the number is decreasing slightly. It has to be mentioned that the available time span of data is rather short to examine this comparison, as students tend to finish their master degree two or three years after the bachelor degree. This also explains the lack of phd theses as the time span is rather short to finish all three degrees.

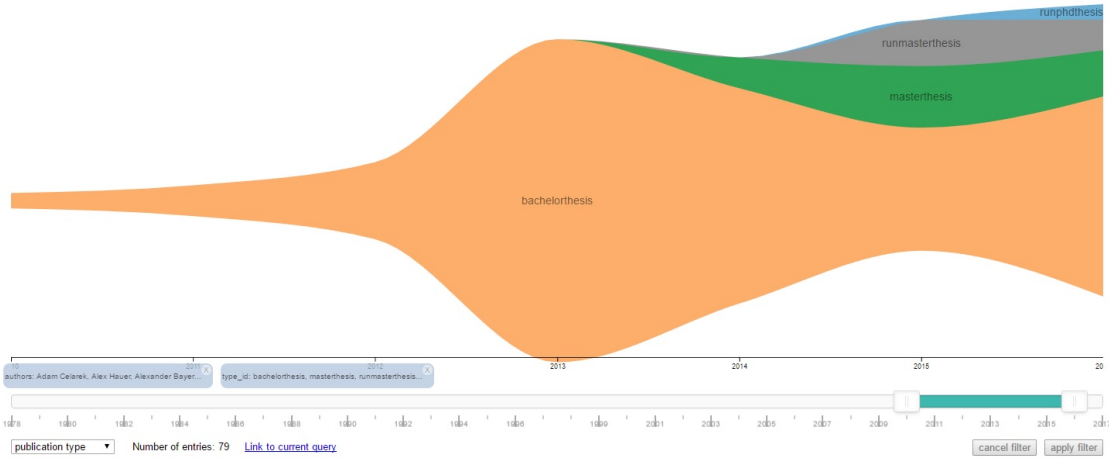


Figure 5.4: Bachelortheses and further theses of higher degree from the same students.

It turns evident in this walkthrough that this user interface can also handle synoptic tasks as the visual encoding provides the visualization to interpret the data as a whole. The visualization is focused around the evolution of attribute occurrences over time in comparison to their sum, the task should also revolve around occurrences over time. In comparison to the first task, this task was more complex to retrieve. The slopes of the layers are an intuitive measure on how the trend of the queried attribute evolves over time.

Also when looking at the author attribute closely, further information can be taken from the attribute evolution over time. As mentioned earlier, in general, students finish their master degree two years after the bachelor degree. This phenomenon can be seen in Figure 5.5, here the selection tool for the filtering was used to highlight authors which finished higher degrees than bachelor. In this case it is easy to find them, the Streamgraph places thicker layer inside the whole layout. Searching around the centerline of the graph for layers helps to find all of these authors.

It is interesting that we are able to find the one outlier, the running phd thesis, in Figure 5.5 by using this technique as well. Around the centerline the only running phd thesis is placed, it is Anna Frühstück. She represents one of the thickest layers, which are placed first, with a written bachelor and master thesis as well as the running phd thesis.

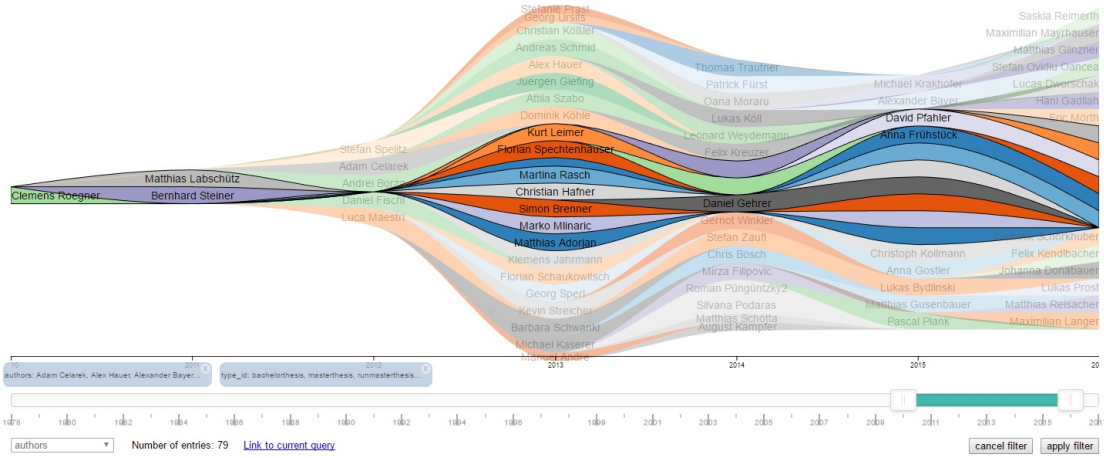


Figure 5.5: Students finish their master degree two years after the bachelor degree.

5.3 Walkthrough: Connectional Task

When studying a phenomenon, an analyst may be interested not only in describing or summarizing the found behavior of a data set but also explain it. Connectional tasks concern with these kinds of questions, find driving forces, that make a phenomenon behave in the way observed. For this walktrough, a closer look is given onto the research areas that Ivan Viola has published in. A filter is set onto the author by clicking the specific author layer in the author view and applying the filter. Then the visualization is switched to the research area view to get the research areas visualized. Specifically the time span from 2013 to 2016 from the research area attribute view, seen in Figure 5.6, is interesting. By using the slider handles the timespan is adjusted to zoom in onto the timespan of interest. When looking at the layers of illustrative visualizations (IllVis) and biological data visualization (BioVis), it can be noticed that the layers mirror each other. This can be noticed as the slope in the whole time span and the inflection points are at the same positions on the horizontal axis of the graph. The connectional task of this section will revolve around the following question: *Is the phenomenon of mirroring layers a coincidence or is there a connection between the two research areas in Violas work?*

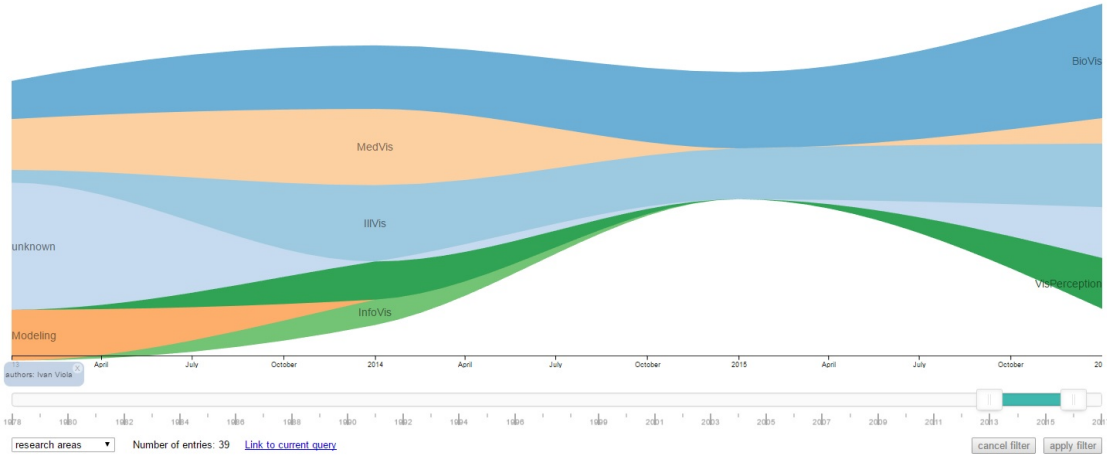


Figure 5.6: The slopes of the red BioVis layer and violet IllVis layer have similar slopes.

To analyze if there is a connection between these two fields in Violas work, it is sufficient to set a filter onto the bigger layer of the two. When filtering for BioVis, the filter shows all publications that are written in this research area. If any other research area layers are visible, this means, that there are publications whose topic is an intersection between both research areas. Figure 5.7 that shows the combined filter which indeed contains multiple layers and shows that a considerable amount of Violas publications revolve around topics that are classified as work in both research areas.

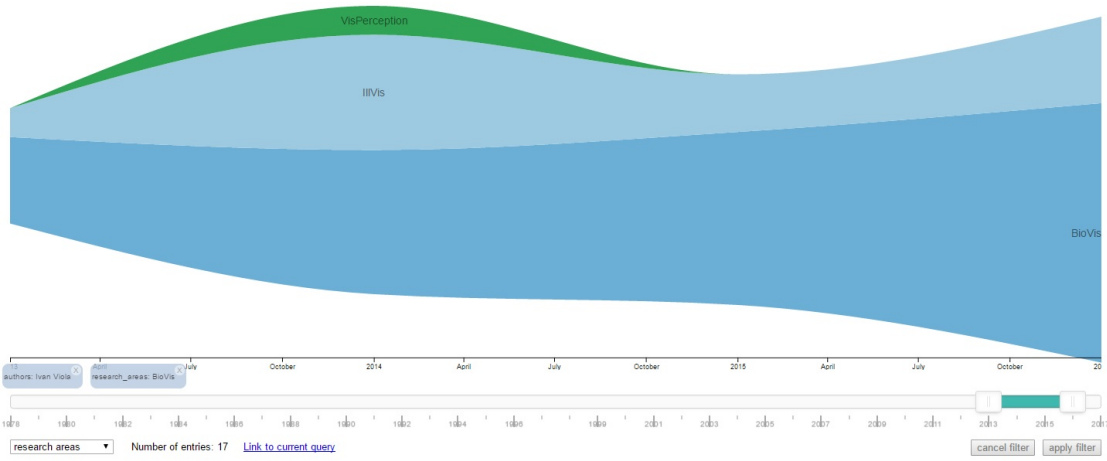


Figure 5.7: Many BioVis publications are also tagged as IllVis research.

With this visualization it is possible to show that there is a connection between the layers. It shows that the IllVis layer is a subset of the BioVis layer, and the publications have multiple research areas assigned to them. The answer to the question therefore is that there is a connection, because Ivan Viola assigned these publications in multiple research areas. However, this question was a very simple connective task. In most cases, especially when looking into less obvious phenomena, it can be impossible to explain behavior only on the basis of the available data. Analysts have to use relevant domain knowledge to reason out pertinent causal links and influence mechanisms [AA06].

This walkthrough shows that the application indeed can find simple connected attributes, which can be done by comparing the slopes and inflection points of layers. This criterion is very intuitive, because when two layers look similar over a longer time span, it indicates, that there is a connection. Using this technique also yields drawbacks, because layers consisting of a low number of publications are very similar to each other. In this case the probability that there is a connection is lower and the criterion of similar looks can be misleading. As mentioned earlier, connectional tasks are hard to answer as the right conclusions often have to be derived from observations with the help of domain knowledge. For the analysis of such tasks a more customizable filter that gives choice whether to build an intersection or a union between selected attribute values of the same attribute would be preferable. Such a change would enable more specific queries with concrete lists of publications that fit the searched topic. Also connectional tasks between attribute values of different attributes are hard to compare as the user has to switch between the attributes to compare the layers. So, a view that gives juxtaposes a selection of attribute views and brushes them consistently would be a valuable extension.

5.4 Result Discussion

Developing the walkthrough gave some valuable insights into the application at hand. During the development it turned out that labeling the layers is a necessity for almost any task. Without the labels it is very cumbersome to get an overview over the query, at every step it was necessary to hover the mouse over the stream to get the label. This work flow was noticeably slower and unnecessarily inconvenient. Further, the performance of certain more complex tasks could benefit if the user had the choice of filtering whether to filter attribute values with an intersection or a union. However, users that are not well informed how set theory works could have problems understanding how to choose the correct operation. As mentioned earlier, in most cases the union is the preferred operation as it always leads to results.

Regarding the chosen tasks, they showed how the Stream I/O performs on tasks from the different levels of the Andrienko model. On the elementary task level it showed that the dynamical list is a valuable tool for analysis as it provides a detailed view onto the individual publications. Nonetheless the visualization can provide some additional information as keyword co-occurrence and collaborations between authors. Synoptic tasks, on the other hand, benefit from the continuous layers of the Streamgraph, as it conveys information over the data as a whole through the silhouette. Descriptive tasks perform very nicely as long as the temporal attribute has a major role in the question at hand. For an analysis on what drives certain phenomena in the data, the application is too simple. For such tasks it can provide some minor insights, however the main contribution has to come from the analyst that provides the domain knowledge.

Besides the benefits mentioned in the walkthrough tasks, these tasks also pointed out a problem with the application. As noted in Section 2.1 the main idea behind stacked graphs such as the Streamgraph is Tufte's macro/micro principle [TGM83]. The goal of the graph is to show many individual time series while also conveying their sum. Since the heights of the individual layers add up to the height of the overall graph, both goals are satisfied at once. This statement is not true for publication attributes that can have multiple values assigned to them. In the 1:N (publication to attribute value) mapping case the height of the overall graph can be higher than the actual number of publications. And so the twin goal is no longer satisfied. To diminish this problem, we decided to normalize the height of the layers so that the added up layers convey the proper sum. Nonetheless this approach results in a trade off, now the relations between individual layers are slightly skewed towards publications with less values per attribute.

At last, the walkthroughs defined some base tasks for a performance evaluation. Additionally, performance was also tested in the key word view, because this view contains the most layers. Evaluating the application revealed the some performance bottlenecks. The multi selection tool as well as interactions, which introduce layer transformations, drop the performance of the application when many layers are visualized. Further, filtering the database also lowers performance, as the application is set back to the start of the pipeline. A data cube structure could potentially shorten the interaction loop.

CHAPTER 6

Conclusion and Future Work

Stream I/O extends the current publication library to add support for exploration and analysis tasks based on the publication database. At start up, the application retrieves the data and normalizes the date attribute. After this cleaning pass, the data is rearranged into a multidimensional data table ordered after attribute and year which sums up the occurrence of an attribute value at time points. This data table serves, as the base for the manipulation that is possible, through the interactive tools. From there the Streamgraph visualization is built using the d3 implementation of stacks that reflect the layout formula of Byron et al. [BW08] for minimization of layer distance and wiggle. To retrieve the final visualization, the stacks are reordered with the inside-out rule suggested by Byron et al. The final Streamgraph is retrieved by interpolating the stacks into a cubic spline that preserves monotonicity in y, using the d3 implementation of Steffens [Ste90] algorithm.

At this stage all interaction tools of the application are available. From this selection the most essential tool is the faceted filtering tool. The implementation allows the user to directly manipulate the Streamgraph to query the database. Together with the attribute selection tool, this enables the exploration of the publication database by navigating along conceptual dimensions that describe the publications. Thus the user is able to view the database from multiple viewpoints and can constrain the query iteratively to arrive at the desired result. All set filters are modular on the attribute level, so they can be altered and removed separately. Furthermore the second dimension of the Streamgraph can be focused by using a slider with two handles. During the whole querying process a dynamical list is also available, which can be expanded to provide the information about the current queried publication while keeping the context to the surrounding publications.

From the results can be noticed that there are multiple benefits this application brings to the publication database interface. Involving the Streamgraph into publication querying, most of all adds support for overview and exploration. Aside from simple queries and queries that search for a priori known, the Stream I/O application provides a more efficient way to query the database. Also while querying, the faceted nature of the application gives insight into every attribute, as it is filtered, by visualizing it in the Streamgraph. Further on, the dynamic list provides a less crowded view onto individual publications that keeps the context to the surrounding publications through its expansion and collapse feature. On the elementary task level the visualization can provide an overview of information like keyword co-occurrence and collaboration, all in relation to time. In addition to this information the visualization also shows the queried subset of publications as a whole through the visual encoding into a horizontal volume. Thus simpler synoptic tasks are also supported by the encoding, as the layer silhouettes provide the information how driving themes evolve in relation to each other over time.

For the future there are a number of enhancements possible. The preprocessing step could be enhanced with automatic or manual keyword cleaning. In the development it got clear that the keywords that users of the database set are very varied, thus hiding the evolution of topics through splitting into a plethora of similar keywords. The data structure of the application could be changed into a data cube structure, which would speed up the interaction cycles of the application. As the workgroup attribute is hierarchic, the color scheme for it could be altered to indicate hierarchic connections. Also the faceted filtering could be extended to let users decide whether they want to filter with an "and-operation" or an "or-operation", thus providing a greater coverage of possible boolean queries. Finally, the Stream I/O could be integrated into the publication website and provide a complement to the existing user interface by enabling further database exploration.

Bibliography

[AA06] Natalia Andrienko and Gennady Andrienko. *Exploratory analysis of spatial and temporal data: a systematic approach*. Springer Science & Business Media, 2006.

[Ama] Amazon. `https://www.amazon.de`. Accessed: 2016-12-6.

[AMST11] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of time-oriented data*. Springer Science & Business Media, 2011.

[Bos] Mike Bostock. D3. data-driven documents. `https://github.com/d3/d3`. Accessed: 2016-11-07.

[Bre] Cynthia Brewer. Colorbrewer2.0. `http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3`. Accessed: 2016-11-21.

[BW08] Lee Byron and Martin Wattenberg. Stacked graphs–geometry & aesthetics. *IEEE transactions on visualization and computer graphics*, 14(6):1245–1252, 2008.

[CLT+11] Weiwei Cui, Shixia Liu, Li Tan, Conglei Shi, Yangqiu Song, Zekai Gao, Huamin Qu, and Xin Tong. Textflow: Towards better understanding of evolving topics in text. *IEEE transactions on visualization and computer graphics*, 17(12):2412–2421, 2011.

[D3.] D3 easy reusable graphs, charts and components. `https://github.com/jamesleesaunders/d3.ez`. Accessed: 2017-02-17.

[Eba] Ebay. `http://www.ebay.at`. Accessed: 2016-12-6.

[Goo] Google images. `https://www.google.de/imghp`. Accessed: 2016-12-6.

[Har00] Robert L Harris. *Information graphics: A comprehensive illustrated reference*. Oxford University Press, 2000.

[HHN00] Susan Havre, Beth Hetzler, and Lucy Nowell. Themeriver: Visualizing theme changes over time. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 115–123. IEEE, 2000.

[IEE] Ieee/iet electronic library. http://ieeexplore.ieee.org/Xplore/home.jsp. Accessed: 2016-12-6.

[IIS+14] Petra Isenberg, Tobias Isenberg, Michael Sedlmair, Jian Chen, and Torsten Möller. Visualization according to research paper keywords. In *Posters at the IEEE Conference on Visualization (VIS)*, 2014.

[Lin] Kevin Lindsey. 2d.js. http://www.kevlindev.com/gui/math/intersection/. Accessed: 2017-02-17.

[LZP+12] Shixia Liu, Michelle X Zhou, Shimei Pan, Yangqiu Song, Weihong Qian, Weijia Cai, and Xiaoxiao Lian. Tiara: Interactive, topic-based visual text summarization and analysis. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(2):25, 2012.

[noU] nouislider, javascript range slider. https://refreshless.com/nouislider/. Accessed: 2017-02-17.

[Req] Require.js a javascript module loader. http://requirejs.org. Accessed: 2017-02-17.

[Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[Ste90] M Steffen. A simple method for monotonic interpolation in one dimension. *Astronomy and Astrophysics*, 239:443, 1990.

[TGM83] Edward R Tufte and PR Graves-Morris. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 1983.

[YSLH03] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM, 2003.