Data visualization in three dimensional space

Datenvisualisierung in dreidimensionalen Raum

Master-Thesis von Jerome [redacted] aus Frankfurt am Main.

Tag der Einreichung:

1. Prüfer: Professor Arjan Kuijper

2. Prüfer:

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Graphisch-Interaktive Systeme

Data visualization in three dimensional space
Datenvisualisierung in dreidimensionalen Raum

Vorgelegte Master-Thesis von Jerome [redacted] aus Frankfurt am Main.

1. Prüfer: Professor Arjan Kuijper
2. Prüfer:

Tag der Einreichung:

Erklärung zur Master-Thesis

gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, den 3. April 2019

(J. [redacted])

Abstract

Two dimensional graphs are ubiquitous and thus have been researched extensively. However they still scale badly as the amount of nodes and edges increases. To reduce cluttering in this thesis an approach is presented that allows the viewer to interactively visualize interesting parts of the graph also making use of three dimensions instead of just two. This way the amount of data is reduced as well as spread out further to simplify an otherwise very complex graph, especially if it is non-planar. This approach applies to multipartite graphs and visualizes subsets that form a unary tree themselves. One example for such graphs are the contents of database tables.

The user selects one subset of the graph after another to determine the unary tree to visualize. Each set of nodes is distributed inside a plane that is orthogonal to the z-axis. These planes are then again spread along the z-axis. This way all edges of the graph are distributed along the z-axis and no longer cluttered in two dimensions. Nodes of one plane are placed near the projections of their connected nodes in the plane below. This layout allows several traditional two dimensional visualization techniques to be applied to the layers as well as additional techniques that are only possible in 3D and still maintains the familiar appearance of a graph closely resembling a tree.

Contents

1 Introduction **7**

2 Related Work **9**

3 Definitions and Notation **13**

4 Approach **15**
4.1 Preparation . . . 16
4.2 Building the Graph . . . 17
4.2.1 Creating the first layer . . . 17
4.2.2 Creating Subsequent Layers . . . 18
4.3 Layout . . . 20
4.3.1 Criteria . . . 20
4.3.2 Candidates . . . 20
4.3.3 Force-Directed Layout . . . 21
4.3.4 Resolving overlap . . . 22
4.4 Highlighting . . . 22
4.4.1 Nodes . . . 23
4.4.2 Edges . . . 26
4.5 Interaction . . . 27
4.5.1 Hover . . . 27
4.5.2 Selection . . . 27
4.5.3 Camera . . . 29

5 Implementation **31**
5.1 Frontend . . . 31
5.1.1 Sidebar . . . 32
5.1.2 Rendering . . . 32
5.1.3 Layout . . . 32

6 Evaluation **35**
6.1 Results . . . 35
6.1.1 Results of the Questions . . . 35
6.1.2 Ratings . . . 35
6.1.3 Problems . . . 36

7 Conclusion **39**
7.1 Evaluation . . . 39
7.1.1 Questions . . . 39
7.1.2 Ratings . . . 39
7.1.3 Problems . . . 41

7.1.4 Surprises . . . 41
7.2 Problems of the Approach . . . 41
7.2.1 Edges . . . 41
7.2.2 Performance . . . 42
7.2.3 Layout . . . 42
7.2.4 Usefulness of Multiple Layers . . . 43
7.2.5 Order of Layers . . . 43
7.3 Comparison to 2D . . . 43
7.3.1 Advantages . . . 43
7.3.2 Disadvantages . . . 43
7.4 Future Work . . . 44
7.4.1 Layout . . . 44
7.4.2 Edges . . . 44
7.4.3 Layers . . . 44

Bibliography **45**

8 Appendix A: Evaluation Form **47**

1 Introduction

Mathematical graphs are ubiquitous and are used in a wide variety of fields to represent data with node-link visualizations probably being the most common visual representation. They are a common tool to make data easier to understand for humans and have therefore been extensively researched in order to adapt them to human perception. This allows humans to identify properties of data at the blink of an eye while at the same time a machine would require complicated algorithms and probably a lot of computational power, making node-link visualizations an important tool in research as well as in in everyday life.

They are however not perfect, especially in terms of scaling. Increasing the size of graph by either adding nodes or edges can render it completely unreadable, especially if the new graph is non-planar. Research has pushed the limits of graphs with techniques like edge bundling or intelligent layouts, but the problem still persists.

Advancements in computational power and rendering enabled yet another approach to solve or at least delay the scaling problems of node-link visualizations. In theory using an additional dimension and moving a graph visualization from two dimensional to three dimensional space should increase readability as the new visualization will be spread out further without necessarily increasing edge lengths dramatically. This could also solve the problem of non-planar graph visualizations as overlapping edges could just pass each other in three dimensional space without ever touching.

This approach however introduced new problems, some of which cannot be solved. In the end all three dimensional visualizations will be projected on a two dimensional surface, either a screen or in the end an eye. This reintroduces the problems of overlap of edges and also makes avoiding overlapping nodes much harder. There are approaches to solve these problems like transparency or making the point of view dependent on interaction, which again requires research on its own.

In conclusion research on node-link visualizations is far from finished, especially in three dimensional space.

This thesis aims to further work on the three dimensional approach, but restricted to multipartite graphs. A multipartite graph (see figure is a graph that can be split into sub graphs with no edges inside these sub graphs but only between them 1.1). This type of graph is very common in the real world as databases for instance can be represented as multipartite graphs with their tables being the sub graphs as long as their tables do not reference themselves. Since the use of databases is very widespread and they might be more tangible for most readers than an abstract multipartite graph, they will serve as examples throughout this thesis.

The goal of this thesis is to develop a technique to visualize generic multipartite graphs in three dimensional space so the viewer can understand the data in respect to several generic metrics like multiplicity. The viewer is not necessarily supposed to see the actual values of these metrics but rather trends, outliers or their absence.

Since this technique works on completely generic multipartite graphs, the resulting graph could be extensively huge. To tackle this issue the graph will be build one sub graph after another, based on the interaction with the viewer, restricting the visible data to only the in-

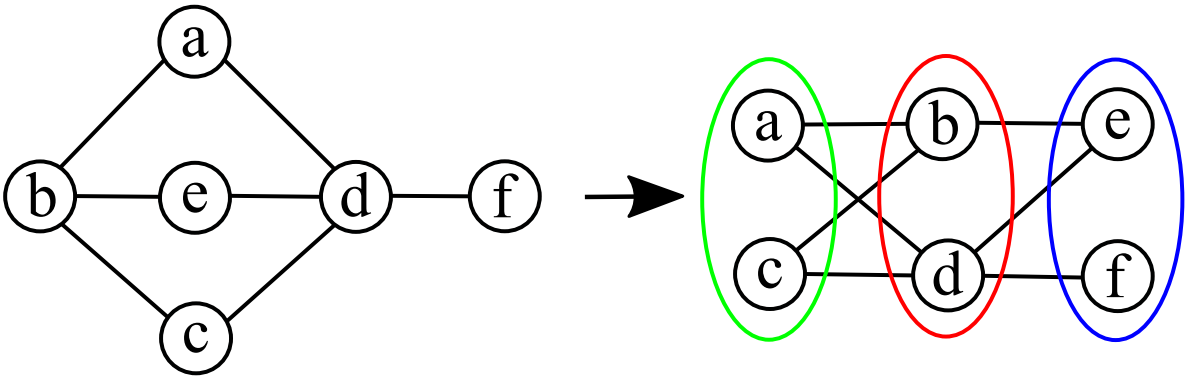

Figure 1.1: Example of a multipartite graph that consists of three sub graphs

teresting parts. To make the resulting visualization even more understandable, the technique will create a layered graph representation with each sub graph being visualized in its own layer. This makes the visualization familiar as it closely resembles two dimensional graph visualizations, when seen from the side, and also allows the application of research for two dimensional node-link visualizations inside of layers. To tackle the problems that arise from projection, the viewer will be enabled to navigate the visualization to see it from multiple perspectives.

This makes use of the viewer's familiarity with two dimensional visualizations, uses techniques from research in two dimensional space, but still make use of the space provided by the third dimension.

2 Related Work

Although the majority of research for graph visualizations has been done for two dimensional visualizations, Kolmogorov et al. investigated visualization of nets in three dimensional space already over half a century ago [KB67].

This long development has lead to a vast variety of very different approaches to work in three dimensions. Orthogonal three dimensional graphs as shown by [BC05] for instance attempt to create a good layout by distributing nodes in a three dimensional grid in space. This approach reduces the complexity of the problem and thus should be taken into consideration when deciding on a layout, but it demands a high amount of space and can appear very un-organic to the viewer due to orthogonal edges.

A less restrictive approach in terms of node positioning are force-directed layouts. They date back to 1963 [Tut63] and use a physical simulation to apply forces to nodes that result from contracting edges and repelling nodes. One major drawback of this approach are problems in terms of performance due to the physical simulation and thus performance has been the main focus of research for two dimensional graph visualizations as well as in three dimensional space. In 2006 Hu [Hu06] introduced a multilevel approach that significantly reduced run time of the algorithm, that is still used in the popular GraphViz library[1] today. In more recent years Toosi and Nikolov [TN16] presented an even faster solution by only taking adjacent nodes into account when calculating forces for the simulation.

Another approach on three dimensional graph visualization is the usage of layers, which is often referred to as 2.5-dimensional as it is technically three dimensional, but still very closely tied to two dimensional representations. These layers could either come from splitting a two dimensional graph representation or by visualizing data that is already layered.

An example for visualization of layered can be found in the work of Brandes et al. [BDS04]. They worked on visualizing metabolic pathways with the goal of allowing an easy comparison of these quite similar structures. The layers in this work represent different but very similar graphs that already existed and are adjusted for comparison using the a common layout of the union of all graphs (see figure 2.1).

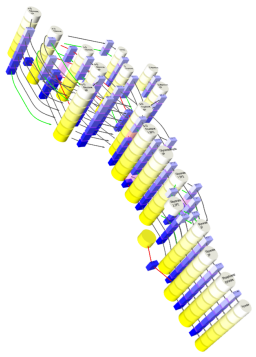

Figure 2.1: Figure taken from [BDS04]

[1] www.graphviz.org

Abello et al. [AHH17] proposed a method to visualize a two dimensional graph using layers by identifying subnets that will become the layers of the graph. These subnets are connected via shared vertices in the layers (see figure 2.2). Their goal is the exploration of a networks topology and to enable this either by inspecting the connection of layers or each layer on its own.

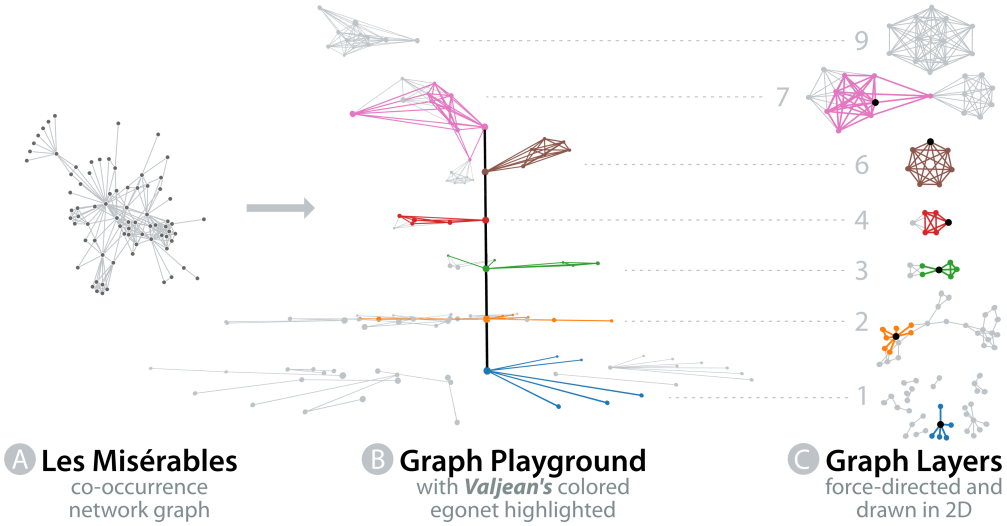

Figure 2.2: Figure taken from [AHH17]

Hong and Nikolov [HN05] introduced a convention on how to visualize directed graphs in three dimensional space. Their goal is to extend the very widely spread method to visualize directed graphs in two dimensions by Sugiyama [STT81] to incorparate a third dimension by splitting the graph into two layers. Later they extended this convention by introducing several alternative algorithms to partition the vertex set into layers [HNT07] (see figure 2.3).

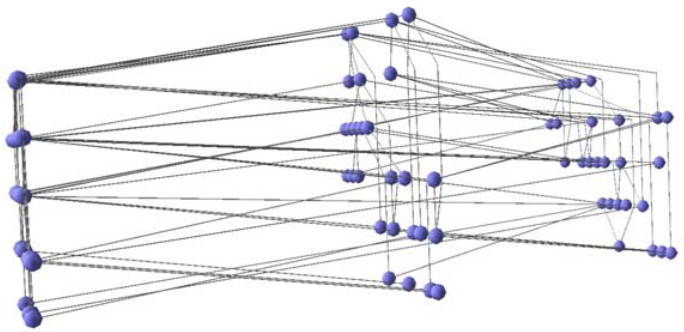

Figure 2.3: Figure taken from [HNT07]

Three dimensional visualizations of graphs often require some sort of interaction to overcome problems like the occlusion or accurate perception of depth due to projection in the rendering process. These problems are often solved by changing the perspective of the viewer interactively.

In their research to enhance two dimensional data plots by incorporating time, Schmidt et al. [SFP+18] propose a three dimensional plot that is specifically designed to be rotated and display the data depending on the view point (see figure 2.4).

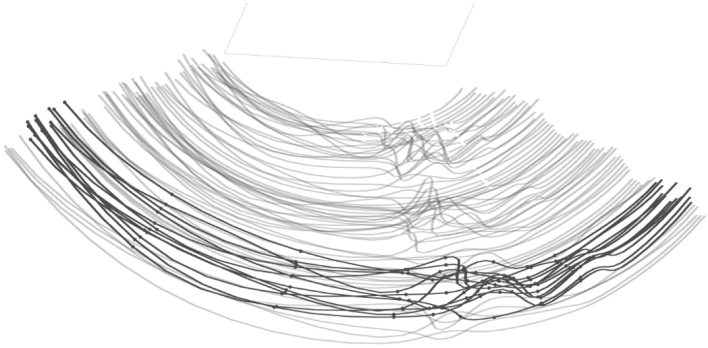

Figure 2.4: Figure taken from [SFP+18]

Newer technologies like augmented and even virtual reality result in further research on how to visualize graphs in three dimensional space and how to navigate in those visualizations. Buschel et al. [BVD19] analyze six variant to visualize edges in augmented reality and evaluate them in a user study, finding no extreme outliers in terms of average task completion times or error rates. However their findings also suggest that users prefer to analyze data from the outside even when using technology like augmented reality. Even though this was not the focus of their work and this thesis uses very different technology, this matters a lot in this context as it emphasizes that data has to be visible from outside the graph. Drogemuller et al. [DCW+18] on the other hand published an evaluation of four navigation techniques, with the conclusion like in traditional visualizations the preferred method of interaction highly depends on the tasks to fulfill.

One approach to help graphs scale better is edge bundling, since it effectively reduces the amount of edges that are visualized. This approach is most effective when a categorization of edges can be found due to the underlying data. However there are approaches to employ generic features of data, like using hierarchy for hierarchical data as seen by Holten [Hol06] (see figure 2.5).

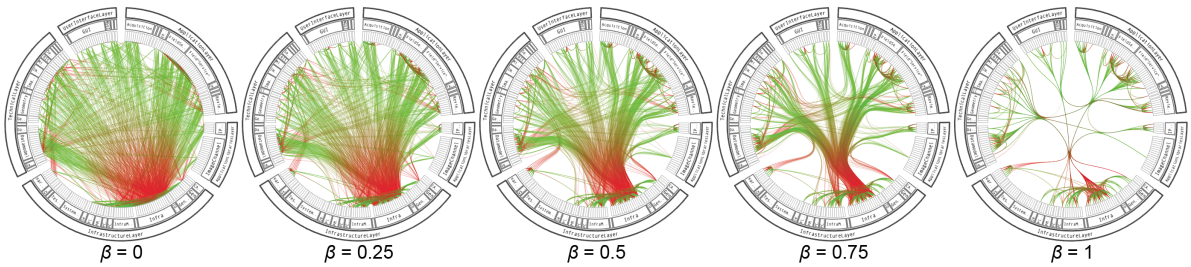

Figure 2.5: Figure taken from [Hol06]

This approach is not limited to two dimensional visualizations as Lambert et al. [LBA10] demonstrate. They apply edge bundling to geographical data mapped to a representation of the earth as a globe (see figure 2.6).

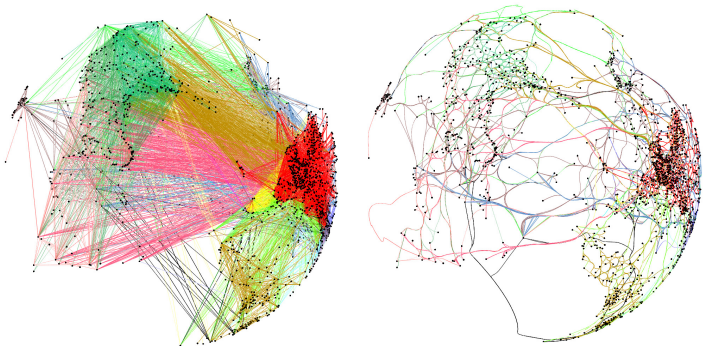

Figure 2.6: Figure taken from [LBA10]

3 Definitions and Notation

Vertices, Nodes A vertex is a data point of the graph and it is visualized as a node.

Connection A connection between sub graphs is the set of edges that connect vertices in between these sub graphs. If this set is empty, there is no connection. In the example of databases, the foreign keys in tables are connections between them.

Layer The subsets of the vertices of the multipartite graph will be called *layers*, as they will be grouped along one axis and distributed in a plane orthogonal to this axis (see figure 3.1).

Up The up direction when looking at the graph is parallel to the axis the layers are spread along. Its direction is away from the first layer (see figure 3.1).

Predecessor The predecessors of a node are all nodes in the graph that are either

- in the previous layer and also connected to that node
- in a layer even further down and connected to a predecessor of the node

Successor The successors of a node are all nodes that this node is a predecessor of.

$\Rightarrow$ This arrow indicates a connection between sub graphs of a multipartite graph or between tables in a database. The direction does not necessarily indicate the direction of that connection and is rather used to indicate the order in which the data is displayed. For example when having two tables, person and skill, with skill referenced by person, the expression “Skill $\Rightarrow$ Person” would indicate that skill is in a layer directly below person and there is connection between them. This does not indicate anything about the direction of the connection between the tables in the database.

$\rightarrow$ This is the equivalent to $\Rightarrow$, but instead of applying to sub graphs or tables, it applies to data points. To extend the previous example, “Writing $\rightarrow$ John” indicates that there is a connection between “John” and “Writing”, but “Writing” is in the layer below “John”, even when “John” might reference “Writing” in the database.

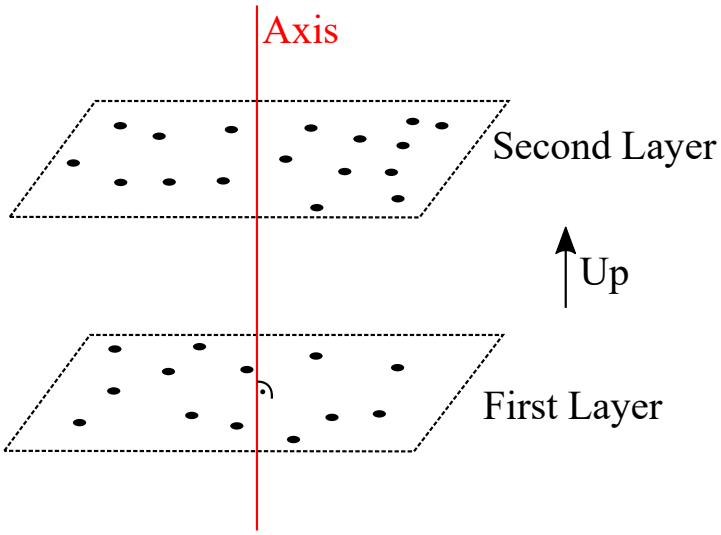

Figure 3.1: Layers in the graph

4 Approach

The approach developed in this thesis is aimed at users that have basic understanding of graph visualizations, while not necessarily being data scientists, and also have already navigated in three dimensional space in several programs, but are not necessarily experts in such tools. So a basic understanding in terms of comprehension of graphs and camera interaction is expected, but the methods used here should still be fairly intuitive and easily accessible.

As already mentioned in 1 the goal of this thesis is to find a better way to visualize a two dimensional graph by reducing the visual clutter that comes along with having many edges and nodes. The basic idea of this thesis is to add more space to the visualization by adding another dimension as well as simply reducing the amount of data visualized by giving the user the ability to select information of interest.

Three dimensional graphs come with their own problems, maybe even the most significant one being the inherent overlap and occlusion that comes along with the projection of the three dimensional visualization on a two dimensional surface. The impact of this problem can be weakened by allowing navigation inside of the graph, however this is a problem on its own as effective navigation with a completely free three dimensional visualization using mouse and keyboard is not very intuitive. There are modern approaches using augmented or virtual reality, but research suggests that even using these very intuitive technologies, users prefer to see a graph visualization only from the outside. This suggests an approach focusing on having a graph that is spread along one axis with a limited radius around it and mostly rely on rotation around that axis for navigation, as it restricts the size of the graph in two dimensions. With this limitation in mind not all types of graphs can be visualized equally, so the focus of this thesis is limited.

This thesis focuses on visualizing *multipartite graphs*. These graphs have the property that they can be partitioned into subsets in such a way that none of the existing edges connects nodes of the same subset. This may sound very restrictive, but in reality these graphs are not rare at all. Database tables for instance can be modeled very easily as multipartite graphs (see figure 4.1) as they rarely have connections of data points inside of a table but rather between tables. Since this is a very common an familiar for the type of graphs in this thesis, database tables are used as examples in this thesis, however the technique developed does not apply exclusively to databases.

Further this thesis does not attempt to visualize the graph as a whole, but rather a subset consisting of said subsets of the multipartite graph that form an unary graph. For databases this would mean that not every connection is shown at once but only specific tables that are either directly or transitively connected and the connections between them. This also means if we have tables A, B and C, and there are connections between all of them, then if all three tables are visualized, there are no direct connections visualized between A and C as this would be not be an unary tree any more (see figure 4.2).

There is obviously no optimal way to choose which subsets to visualize, so the graph has to build in interaction with the user. They choose one subset of interest after another and as such build the graph that actually matters to them, leaving out unnecessary parts.

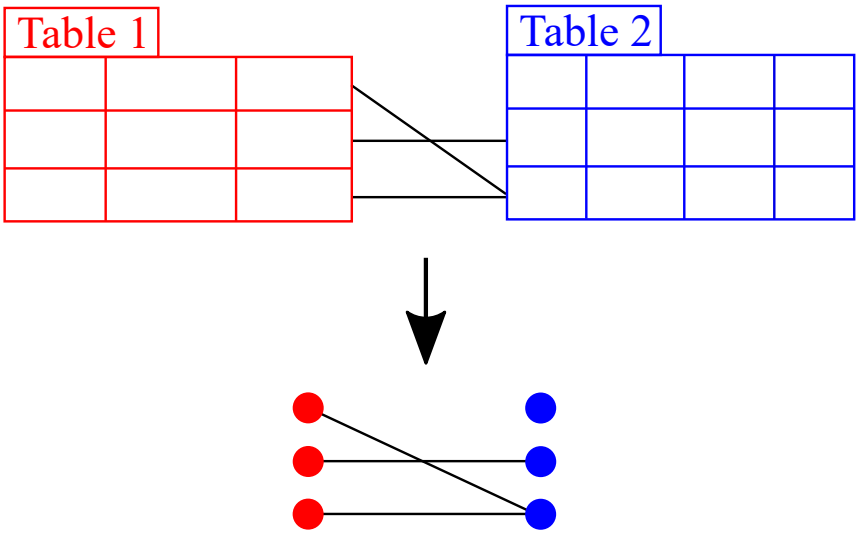

Figure 4.1: Relation between databases and multipartite graphs

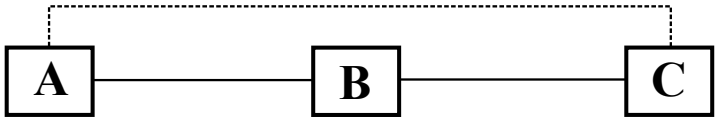

Figure 4.2: The link between *A* and C will not be visualized if *B* is visualized, is it would not be an unary tree anymore.

This chapter is structured by the steps of the technique in chronological order. That means first the preprocessing steps necessary to prepare the graph are described and after that the workflow a user would be guided through.

4.1 Preparation

Before building the actual graph the subsets of the two dimensional graph have to be identified, that make it a multipartite graph. Since having a multipartite graph is a core requirement of this thesis, it is assumed that this step has already taken place. Identifying a multipartite graph is not in the scope of this thesis.

In order to build the graph a reachability graph has to be build. This graph indicates which subset of our two dimensional graph is connected to which. Staying with our database example, we have to build a a graph depicting which tables are connected via foreign keys. Each table becomes a node in our reachability graph and each edge represents a foreign key. Since we are building a unary tree using these tables, the direction of the connection does not matter. The graph does only express the existence of connections, not their directions.

The actual data structure used for an implementation of this reachability graph does not matter. Using something else like a matrix is completely sufficient, as long as some sort of path finding algorithm can be applied later on. For easier explanation in this thesis a graph has been chosen and will be used in the future for consistency, but keep in mind that this is not a strict requirement.

Since this reachability graph does only contain direct and no transitive connections between the subsets and operates by ignores directions, building it has a complexity of $O(n)$.

4.2 Building the Graph

When interactively building the graph two different approaches have to be taken, differentiating between the first layer and subsequent layers. The creation of the first layer is free while the visualization of subsequent layers highly depends on the previous layer.

4.2.1 Creating the first layer

Layout

The first step is the user choosing a graph subset or database table of interest to form the base layer of the graph. This will influence the layout as well as other properties of all following layers. Since this subset could contain all kinds of data, there is no way to determine the best way to layout this and a random distribution is applied.

In this thesis the plane that contains all points of a layer is positioned parallel to the xy-plane and the planes of all layers are spread along the z-axis. This choice is arbitrary and completely different planes and axes could be chosen, but the planes need to be parallel and their points should be centered along one line that is orthogonal to the layers. This way all points can be found around one center line.

The points of the first layer are spread arbitrarily in the first layer, but for special cases where the data is known, a more specialized solution can be applied. For geographic data for instance a map can be rendered inside the layer's plane and the data points can be distributed according to their actual positions on this map.

If no special distribution is applied, a resolution of collisions might be necessary. In thesis this is done by applying a force-directed approach as described in 4.3.4.

Color

For further differentiation of points and also association between points of the first layer and their successors, the points are colorized. Since the amount of points is arbitrary, a fixed list of 'good' colors is not a sufficient.

The first solution that comes to mind to color n points is to generate n evenly distributed numbers in $[0..2^{24}]$ (assuming 24 bit RBG colors are used to display the points) and then mapping these number to their RGB values. This however does not work well as it does not only sample different color values but also different brightness and saturation and generates also different shades of gray, when many values are generated, which are hard to distinguish.

To have colors that are more easy to differentiate, sampling the HSL is a much better option. This way fixed values for saturation and luminosity can be chosen and only the hue are sampled (see figure 4.3). This results in a much more natural and intuitive distribution of colors and leaves saturation and luminosity open to indicate other properties of data.

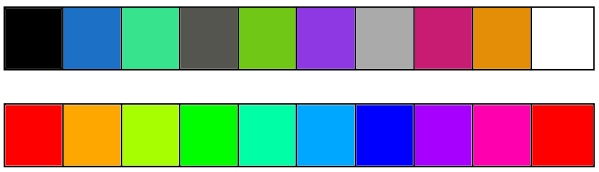

Figure 4.3: Sampling ten evenly distributed values from RGB (top) and from HSL (bottom) with fixed saturation and luminosity

There is one important property to keep in mind though: the hue in HSL is defined circular (see figure 4.3), so the samples have to be chosen in an half-open interval, otherwise the first and last value would be the same.

4.2.2 Creating Subsequent Layers

After creating the first layer, the user hast to select the next sub graph or database table of interest. This is where the reachability graph becomes important. Using a breadth-first search all directly or transitively connected sub graphs or tables are determined and presented the user to choose from. Depending on that choice the connections between these new vertices and the vertices of the lower layer have to be determined. This is obviously trivial for direct connections, but some sort of path finding is necessary for transitive cases. These connection paths then have to be shortened to a single-step transitive connection.

For example for three database tables A, B and C, with direct connections via foreign keys from A to B and B to C, the path from A to C would be $A \rightarrow B \rightarrow C$. The connection $A \rightarrow C$ could be determined through joins of A, B and C. Given vertices $v_a \in A$ and $v_c \in C$ there might be several vertices in B that connect them, resulting in duplicate edges $v_a \rightarrow v_c$. These duplicates are removed and instead the edge $v_a \rightarrow v_c$ their number is saved as *multiplicity* of the edge (see figure 4.4).

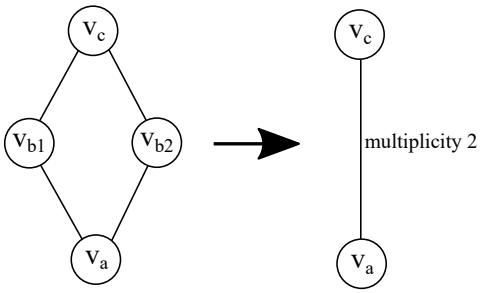

Figure 4.4: Edge multiplicity of transitive connections

Whenever a new layer is to be added, only the previous layer and and the data of the new layer are taken into account. This is due to performance concerns of the layout (further discussed in 4.3) in combination with the speed requirements due to the interactive approach of this thesis.

After positioning the vertices of the new layer according to 4.3, edges are drawn between connected points. They are not being bundled as this would require some attribute to group them by, but since the data is generic, no such attribute can be computed automatically. To still make the graph visually more appealing and give the edges more expressiveness as well, they are curved. They are visualized as cubic bezier curves, using the nodes they are connecting as end points. The other two vertices of the control polygon are directly above or below those nodes and their distance from those nodes depends on their multiplicity (see figure 4.5). Usually in this thesis multiplicity of nodes always only depends on the incoming edges from below. In this case however for the lower nodes the number of edges leaving them to the top is meant. This results in a curvature of the edge that sort of 'bundles' edges at nodes with higher multiplicity and leaves edges between nodes with both multiplicities of one as a straight line.

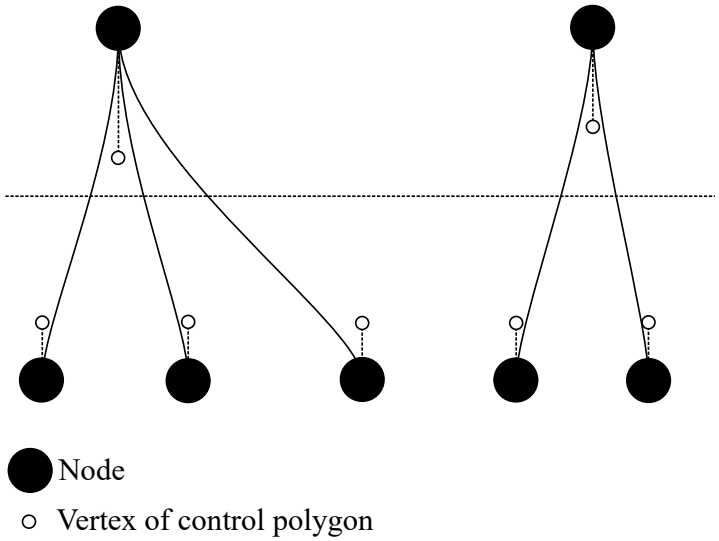

Figure 4.5: Curved edges with control polygon (nodes are part of the control polygon of their connected edges)

Edges are drawn using the hue of the node in the lower layer that they originate from to increase the visual connection between points and their predecessors. The saturation of the edge is used to indicate its multiplicity. For that purpose the maximum multiplicity of all edges in that layer and a minimum saturation are determined. Assuming saturation is expressed in the interval [0..1] it is calculated as seen in 4.1. This guarantees a minimal amount of saturation to still be able to distinguish colors while also indicating multiplicity of the edge.

$$\begin{aligned} saturation(multiplicity) = & saturation_{\min} + (1 - saturation_{\min}) \cdot \\ & (multiplicity_{\max} - multiplicity) \end{aligned} \tag{4.1}$$

The color of the vertices of a new layer is determined by blending the colors of all incoming edges from below in RBG color space as depicted in 4.2. This visualizes the connections of a layer to the lowest layer and allows to see outliers. The less 'mixed' a color appears the less different points on the lowest level it is associated with.

$$\begin{aligned} red &= \frac{\sum_{edge \in lower\_edges} edge.red \cdot edge.multiplicity}{\sum_{edge \in lower\_edges} edge.multiplicity} \\ green &= \frac{\sum_{edge \in lower\_edges} edge.green \cdot edge.multiplicity}{\sum_{edge \in lower\_edges} edge.multiplicity} \\ blue &= \frac{\sum_{edge \in lower\_edges} edge.blue \cdot edge.multiplicity}{\sum_{edge \in lower\_edges} edge.multiplicity} \end{aligned} \tag{4.2}$$

Nodes are neither visualized transparent nor with low saturation to make them clearly stand out from their edges when looking from above.

4.3 Layout

When adding a new layer of data, a layout has to be found to distribute the nodes in the layer's plane. Since layers are two dimensional, they do enable the use of two dimensional layout algorithms to some degree or at least slightly adapted versions of them.

4.3.1 Criteria

When choosing a layout, several criteria have to be met:

- The nodes must not overlap. Overlapping nodes make it exceptionally hard to distinguish nodes as well as the edges they are connected to.
- Nodes of a new layer should be close to the projection of the nodes from the lower layer they are connected to. This reduces the lengths of edges, making it easier to follow them visually, and also create a more intuitively understandable connection between nodes, as nodes that belong together are close to each other.
- The creation of the layout should not take longer than a few seconds. This approach is based on interactivity, so displaying data should ideally happen without visible delay, but this might not be possible depending on the scale of the graph.

4.3.2 Candidates

The first idea that comes to mind when trying to express properties of data through the layout is mapping those properties onto the axes and position nodes accordingly. This approach might work for special cases and would provide the maximum amount of expressiveness while reducing the computational effort to a minimum, but for the general case this approach does not work. It is very prone to overlapping nodes or even creating heaps of nodes if the data is not evenly distributed in the chosen properties. Since the goal of this thesis is to create an approach that is applicable to generic data, this layout is not considered the right choice.

One thing to keep in mind when choosing the layout is to minimize the delay caused by applying it. One potential solution that comes to mind when trying achieve this is pre-computing layouts once and using those later. This option does not seem feasible because it would be very expensive in terms of computation as well as memory, since each sub graph can be combined with each other sub graph in two directions, and this for potentially thousands or ten-thousands of connections between them. In addition to the memory and computational requirements, this approach would also imply that an implementation has the time to pre-compute all layouts without impact on the user interaction. This limits the amount of potential candidates.

However there is still a plethora of two dimensional layouts to choose from, but ideally an algorithm that allows to avoid or resolve overlap and optimizes the positions of nodes locally around the projections of their connected nodes should fulfill the defined criteria best. A force-directed layout fits these criteria quite well and hence is used in this thesis.

4.3.3 Force-Directed Layout

A force-directed layout works using a physics based simulation by applying forces to each node and moving it accordingly. Nodes repel each other while edges try to contract and pull nodes together. This results in a graph where groups of strongly connected nodes form cluster-like structures that are often clearly separated from each other.

Application in this Approach

Ideally one might want to apply the layout to all nodes whenever a new layer is added. This would probably provide the best results in terms of expressiveness of the layout, but is computationally complex. In order to scale better, only the new layer and the previous layer will be taken into account when a new layer is added. Since the layout is applied with each new layer, the previous layer is considered as properly laid out and will stay fixed in position.

The algorithm used here is targeted at two dimensional visualizations. To apply it to two layers at once, the lower layer is projected to the upper layer and later removed. Disconnected nodes in the new layer that have no connection to the lower layer, have to be treated separately either by applying an additional force to prevent them from being pushed too far away since no edge is holding them back or completely removing them from the graph since they might not contribute anyways. This decision however depends on the data.

Finding a good starting configuration is crucial for local optimizations like the force-directed layout. This can be especially hard if it is supposed to be applied to generic data, where no further information is known. This problem here is solved by placing each node at the averaged position of its connected nodes in the lower layer (see figure 4.6). When calculating this starting position however, the multiplicity of the edges has to be taken into account. That means a node a that is connected to nodes b and c would be exactly in the middle of b and c if the connections to them had the same multiplicity. If the multiplicity of the edge between a and b is higher than the multiplicity of the edge between a and c, then the starting position of a would be closer to b (see figure 4.7).

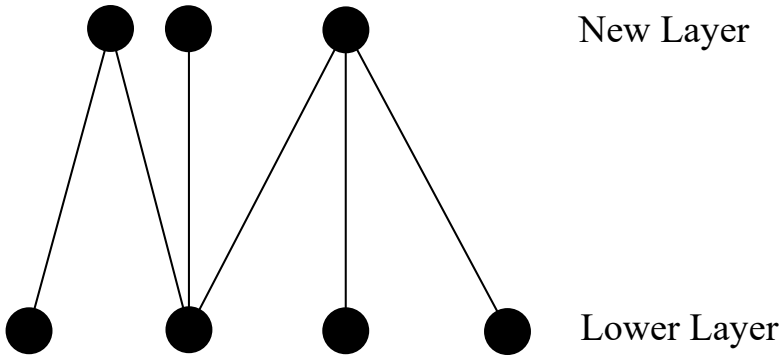

Figure 4.6: Choosing starting positions

This starting configuration also has the effect that nodes with higher multiplicity automatically are moved to the middle of the layer plane, giving the layout additional informational value.

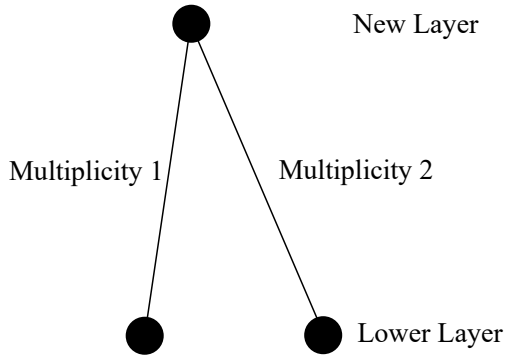

Figure 4.7: Choosing starting positions taking edge multiplicity into account

Changes to the Algorithm

The previously described approach to finding starting positions is simple, intuitive and effective, but creates a new problem: The forces between nodes should be inversely proportional to the distance of the nodes. The starting configuration creates a lot of cases, where nodes start from exactly the same position, thus leading to a division by zero. One can apply a very small offset in a random direction to all affected nodes to avoid divison by zero, but the resulting forces would still be incredibly high. This leads to extreme inital movements into random directions, rendering the layout useless even after contracting the edges. To counteract this problem, the forces applied can be limited to a certain amount by either making a hard cut in the force function of applying a smooth function like a modified logistic function (see figure 4.8).

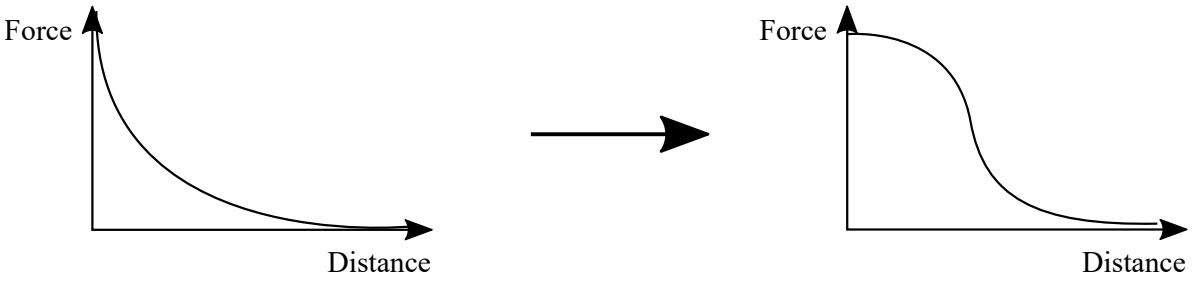

Figure 4.8: Change of force functions

For this approach however a slightly less strict version of the standard force-directed layout can be used. Since the end position of a node should not vary too much from its starting position, and the main goal is to avoid overlap of nodes, not all nodes have to apply forces on each other. Only a very restricted neighborhood has to be taken into account.

4.3.4 Resolving overlap

4.4 Highlighting

The most obvious property of nodes to visualize is their connectivity, which is highlighted using edges as well as colors as described in 4.2.2. Aside from that there are three other metrics that will be highlighted:

- The **multiplicity** of an **edge** is the number of edges it visualizes. For a simple edge this multiplicity is one, but if there are edges $a \to b \to c$ and $a \to d \to c$ then the multiplicity of $a \to c$ would be two.
- The **multiplicity** of a **node** is the number of incoming edges from the previous layer that end in this node. It takes the multiplicities of the incoming edges into account by using them in the summation.
  For example if there are edges $a \to b \to c$ and $a \to d \to c$ then c will have a multiplicity of two, even though only one edge might be visualized when displaying $a \to c$.
- The **co-occurrence** of a node in regard ot another node expresses how strongly they are connected by the data of the previous layer. That means the co-occurrence of node a in regard to a node b is the number of paths $a \leftarrow c \to b$. It also takes the multiplicity of of edges between those vertices into account by summing them up.

The focus of this thesis lies with the visualization of generic data, so generic metrics that still express important information had to be chosen. With these metrics it is possible to measure for generic data how important single points of data are and how strongly nodes between different or inside the same sub graph are. They should also enable the viewer to answer simple questions about a multipartite graph and identify points of interest. Since the absolute number of nodes and edges of the graph arbitrary, the metrics need to be normalized or otherwise their visualization might not be understandable.

These metrics can be visualized in several different ways, some of which incorporate the additional space offered the use of three dimensional space, while many are also usable in two dimensional space. Not all of the following visualization techniques have proven to be useful, but they will still be mentioned as they seemed promising beforehand.

4.4.1 Highlighting Nodes

Offset One of the visualization approaches making use of the third dimension are offsets. One can apply offsets in the up direction to express the magnitude of a metric, resulting in a visualization similar to a scatter plot. This approach might be reasonable if values of the visualized metric slowly but steadily rise towards the center of the layer, but in general it is not very helpful. First of all there is the problem of nodes further away from the layer center occluding nodes further inside, if the the values do not rise towards the center. Another problem is the perception of depth if values rise too quickly from the outside to the center as the visual clue of partial occlusion is no longer present. It might be hard to tell for neighboring nodes which one is in the front and which is further away (see figure 4.9).

A benefit and problem at the same time of this approach however is the independence of the edges to the previous layer. One benefit of this layered visualization approach is the separation of nodes and edges. To avoid mixing these two when applying the offset, nodes need to hover above their edges, which still end in the position of the node without the offset (see figure 4.10). This is beneficial on the one hand as node metrics can still be visualized when the edges are completely entangled and are not recognizable anymore, but on the other hand it becomes much harder to to see the connection of a node with offset to its connected nodes in the previous layer.

Lastly it is worth mentioning that using offset only works if there are enough nodes

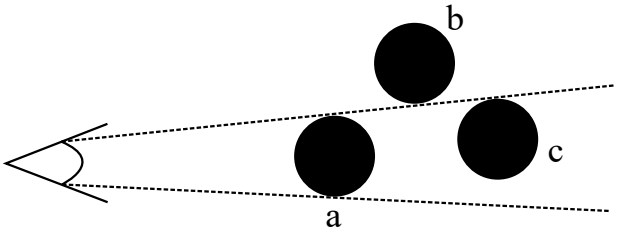

Figure 4.9: Node *a* hides node *c* and the difference of depth of node *a* and b is hard to guess due to lack of partial occlusion.

to apply them to. If there for instance only three nodes, then the perception of height per node is very view dependent. The closer the nodes start to resemble a continuous surface, maybe with a peak, the easier it becomes to see their relative height differences. Due to these problems, this approach is not pursued any further.

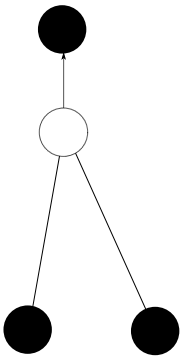

Figure 4.10: Edges are not affected when applying the offset.

Pillars Two of the problems of the offset, the lack of partial occlusion with the resulting problems with depth perception as well as the distance to the edges, can be solved by stretching the points until their lower end reaches their edges again. Hence the logical successor to the use of offset is the visualization using pillars. They are very commonly used as bar charts in two dimensional and sometimes even three dimensional space and thus should be very intuitive to understand for many viewers.
The usage of pillars avoids the problem of depth perception as pillars in the front will always occlude at least a part of the pillars in the back (see figure 4.11). However this increases the problem of completely hiding other nodes, which can be mitigated by allowing the viewer to change their perspective, for example by rotating around the graph or to tilt the view. This still remains a problem unless the height of the pillars increases monotonically towards the center as this allows to resolve all occlusion problems by changing the perspective and point of view as described before.

Pillars also benefit of the separation of the potential clutter of underlying edges, as they visually rise above them while in contrast to nodes with offset still being connected to their edges. The problem of not being able to tell the relative heights of nodes if not enough data points are available is mitigated to some degree, especially if the pillars are close to each other, but otherwise it is still view dependent. For a low number of nodes there are better visualizations of metrics.

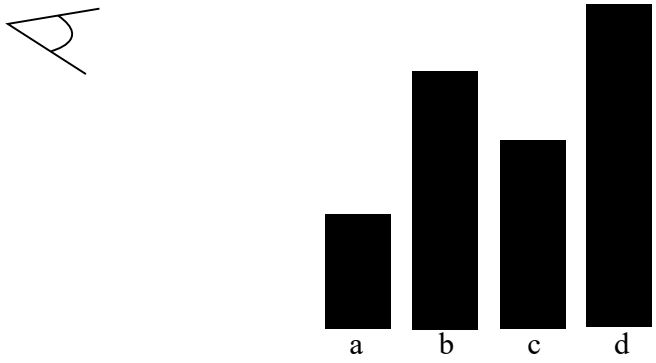

Figure 4.11: Node *a* is clearly in front of all other nodes to the viewer, but *c* is not visible.

Size Still in the spirit of changing the changing the size of the nodes, one can simply scale inside of their two dimensional plane. Similar to the pillars this is a very common approach and should be easy to interpret by the viewer, but suffers less from occlusion problems. This approach however is better suited for fewer nodes as it relies less on creating the visual approximation of a continuous surface, but rather displays discrete values. It also does not suffer from the problem of disconnecting nodes from their edges as they are not moved, retaining the visual clues representing connection to the previous layers.

The greatest drawback of this approach however are the implications for the layout. It has to be recalculated in order to avoid overlapping nodes, which might take a while to compute, resulting in latency for the viewer. There will also be greater gaps between bigger nodes that might be filled by the force-directed layout using smaller nodes, which results in the position of a point losing its informative value.

Color Coding A very different approach to those previously mentioned is simple color coding. The normalized values of the metric can be mapped to a color gradient, for instance from green for lower values to red for higher values. This is yet again rather intuitive to interpret for many viewers and does also obviously not affect occlusion or the layout in any way.

However since the color values of the nodes are changed, they lose one visual clue representing a connection to previous layers. As mentioned in 4.2.2 the colors of nodes are mixtures of all their parents further below in the graph, but this information will be lost using this approach.

There is also the problem of humans only being able to distinguish a limited amount of colors. That means very close values of the visualized metric might be indistinguishable using this approach, but since highlighting in this graph is rather used to find extreme outliers or determine their absence instead of making exact values visible, this problem is not considered crucial in this thesis.

Color Components To mitigate the problem of losing the visual clue indicating connection to the first layer as seen in color coding, one can take an eased approach by not changing the color completely but only in components. The hue should still remain the same as a change here would be perceived as a complete change of color. Luminosity, saturation or a combination of both are all good candidates to visualize the normalized values of a metric.

However there should be a lower bound on both values as lower values otherwise would only become gray or black and the there would be the same problem as in color coding. Luminosity also needs an upper limit to prevent nodes from becoming just white and ignoring their hue and saturation.
While changing color components has the benefit of prevailing the hue in comparison to color coding, it increases the chance of having indistinguishable colors due to the limited scales of luminosity and saturation used.

Combinations Some of the previously mentioned highlighting methods can be combined to either visualize two metrics at once or make the values of one metric even more distinct. You could for example combine an approach based on colors with a visualization that changes the shape of the nodes, or even combine changing the size of the nodes with visualizing them as pillars.
While this is technically possible and might be helpful to find out about correlations, it is in general not advised to do so as it may overload the visualization and over strain the viewer. In addition to that there is the problem that the chosen metrics are already correlated. A node with a high multiplicity also has a high chance to have a high co-occurrence in regard to other nodes as it occurs often in general. So the benefit of visualizing them at once may be not worth the cost of over complicating the graph visualization. It should be up to the user to decide, whether or not to use both, as there might be outliers in the data, that are interesting.

4.4.2 Highlighting Edges

Thickness The first idea that comes to mind to visualize the multiplicity of an edge is changing its thickness, so a line that represents many edges would be thicker or wider than a line representing a single edge. This has however one significant drawback as it further clutters the space used by the edges between two layers and also increases occlusion. This approach would work for very few lines, but it has huge problems when it comes to scaling. One could try to mitigate that problem by defining a range of thicknesses that the normalized metric is mapped on, thus limiting the amount of clutter that each edge can add. However this range must be fairly limited, since a line of ten pixels for example is already really wide, leaving only nine increments of thicknesses to express values of the metric.
Due to the significant problems of cluttering, occlusion and expressiveness this approach is not further pursued.

Transparency Another obvious choice to visualize edge multiplicity would be transparency. Visualizing the edge with the most multiplicity opaque and the least multiplicity as barely visible seems to be a reasonable approach. This way not only edge multiplicity would be displayed but also the problem of cluttering of edges would be mitigated.
However there are also significant drawbacks. A good blending function has to be found if there are many edges overlapping otherwise one ends up with a seemingly opaque mass that really does not visualize anything anymore. The main issue though is not the blend function, but rather the way the colors of the edges is determined. They inherit their color from the nodes in the lower layers, which again originally inherited theirs from the first layer. The colors of the nodes in the first layer are determined by sampling

the HSL color space (as described in 4.2.2) depending on the number of nodes. This results in colors with very different contrast to the background, which then again may appear more or less clearly when transparency is applied (see figure 4.12). This warps the perception of transparency significantly and makes this approach unusable in this case.

Figure 4.12: The effect of transparency to lines with different hues. Saturation and Luminosity is equal for all lines, the top lines are both opaque, while both lower lines have the same alpha value.

Visibility The easiest and most radical approach to visualize edge multiplicity is by allowing the viewer to define a range of values and only visualizing edges that have multiplicity in that range. This should not be the preferred method as it moves the responsibility to create a valid visualization to the viewer, but it is very effective, expressive and also very much helps with the problem of cluttered edges.

4.5 Interaction

Aside from building the graph and applying metrics interaction is used in several ways to pursue very goals, such as changing the view or show underlying data.

4.5.1 Hover

This graph visualization has the goal to give a rough overview over the given data and to identify outliers or their absence in regard to certain metrics. Sometimes however a viewer might want to see the actual data of a node, for example to identify what data an outlier actually represents. This can be done by allowing the viewer to hover the cursor above nodes to see their underlying data. How easily this can be done of course highly dependents on perspective and occlusion, which emphasizes why the thoughts about these topics in 4.4 were so important.

How and what data is visualized when hovering over a node is of course very implementation dependent. For databases for example all values of the row that the node represents could be visualized in a tool tip, allowing the viewer to see all data, or a filter could be applied.

4.5.2 Selection

Selection of data can be a very powerful tool to explore certain parts of a graph as they can be made visually distinguishable from rest of the graph. Since the data of this thesis is mostly stored in the nodes and the edges only merely represent the connections between these data points, only nodes will be directly selectable. However since node-link visualizations are

all about nodes and their connections, edges and nodes directly or transitively connected to selected nodes will also be selected. In combination with the ability to directly select multiple nodes at once allows to investigate sub graphs of the visualized graph. Depending on the way selection is visualized this can either be seen in parallel to the rest of the graph to allow comparison or it can be visualized exclusively.

Nodes

Of the aforementioned highlighting methods for nodes only the approaches based on colors seem appropriate to visualize selections. Offsets, pillars and sizes share the property that they would all be very arbitrary in their extent, as they are usually used to express relative amounts of something, which makes very limited sense for the binary selection property. Offsets would also have the effect of disconnecting nodes from their links as discussed in 4.4.1 and applying an offset to nodes in the lower layers would be even worse as they would be moved into the edges to their next layer. The last problem also applies to pillars. Size changes do not have this problem, but in addition to their arbitrary values they have the problem of making a re-layout necessary. Since connections often spread through all layers, this would mean the layout of the complete graph would need to be recomputed, which might be very time inefficient.

Yet another way to visualize selection would of course be visibility. It is a binary value by default and has the potential to clean up the graph significantly, which would make it a really great candidate. However this would make selection of multiple nodes impossible, as other nodes would be either invisible or already selected indirectly after selecting the first one. Due to this significant drawback, this option is ruled out in this thesis, although it would even be the best option if one is willing to give up allowing multiple selections.

This only leaves changing the colors or part of the colors as the only options for visualizing selection of nodes. These color changes can be applied very differently to selected or non-selected nodes.

- **Selected** nodes could be highlighted by increasing their saturation and/or their luminosity, but this obviously requires them to not be at their maximum already. The colors could also be changed completely to a color that is not used yet, like black or white. This would probably be the better option than changing only elements of the color, since the contrast to surrounding nodes would be much larger, but this would also mean that they lose the visual clue indicating their origin. In order to retain this visual connection as well as get the increased contrast one can partially change the color of the nodes. For example for a visualization of nodes as circles that are rotated to face the viewer one could change the color of the outer ring to a color that has a strong contrast to the background while also changing elements of the colors of non-selected nodes.

- **Non-selected** nodes can either be visualized by changing their color completely to a single color or changing elements of their colors. The last option provides the benefit of retaining the visual connection to each nodes predecessors and also keeping the general appearance of the graph more closely to its original state.

With both of these options combined, partially changing the color of the selected nodes, while also changing elements of the colors of the non-selected nodes, selection of nodes becomes very easy to identify.

Edges

Although edges are not selected directly, they will still be highlighted if they connect selected nodes. However since they can not be selected directly this also means that they do not need to be visible to enable multiple selections, which makes visibility the best candidate to visualize selection of edges. This is especially true since a big problem of node-link visualizations is the amount of edges they contain. With this option visually understanding connections becomes a lot more straightforward.

Search and Filter

Selection is a very useful feature, but it the more nodes are in the graph, the harder it becomes to select a single one. To assist in selecting a feature or make the graph more manageable in general a filter for nodes and/or edges might be helpful. They can be used to filter depending on the value of graph elements for a metric like multiplicity, so elements outside a certain interval will be removed from the visualization.

This however might not be enough to find certain nodes, so a search functionality might be necessary. The implementation of this however depends very much of the visualized data, so for databases one might obviously want to filter depending on the layer, using the columns in the table that it represents.

4.5.3 Camera

Perspective plays an incredibly important role in three dimensional node-link visualizations. Since this thesis is written with arbitrary data in mind, an optimal point of view is never guaranteed to exist and would still be hard to automatically find if it existed. Hence an interactive camera is necessary to allow the viewer to adapt the perspective to the problem at hand.

Since the visualization resembles a pillar, with nodes and edges alternating an axis that they are centered around and a layer of nodes that has important visualization of metrics four ways to interact with the camera seem necessary:

1. Rotation around the axis of the pillar: It is necessary to rotate around the data to avoid not being able to see due to occlusion.

2. Moving up and down along it: In theory this pillar could grow endlessly. In practice there are limits to how many layers are useful to visualize, but those limits are unknown, so a way to navigate along this axis it necessary.

3. Tilting the camera up and down: In order to see nodes from different view, tilting is necessary, especially when one has to see differences in height. This is the case when visualizations using offset or pillars are used.

5 Implementation

The approach described in this thesis has been implemented as a prototype. It is based on web technology and uses a database as the graph to visualize. The application is written with arbitrary databases in mind and uses its metadata for queries. This way several databases could be used to test if the approach is truly independent on the underlying data.

The preparation phase of the workflow (see 4.1), consisting of building the reachability graph, is executed when the backend is started as it only needs to run once. It analyses the database at hand by extracting the names of all tables as well as their primary and foreign keys to determine connections between them. This allows the use of pretty much any SQL database (connections to MySql and Sqlite have been implemented) as long as this metadata has been set properly.

5.1 Frontend

The frontend is implemented as a website and consists of two parts (see figure 5.1):

- A sidebar for the user interface
- A render canvas, where the data visualization will be displayed.

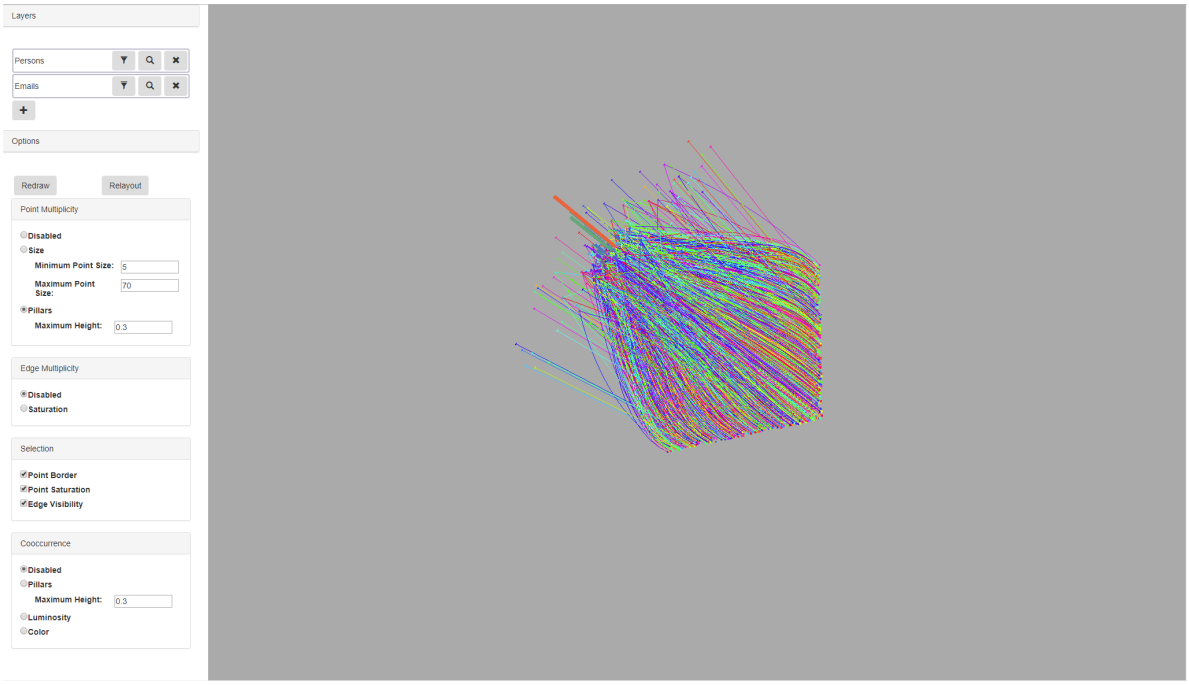

Figure 5.1: User interface of the prototype

5.1.1 Sidebar

The sidebar has a part displaying all layers that are currently visualized, allows to add and remove layers and to apply actions to each layer individually (see figure 5.2). As described in 4.5 it allows to filter nodes and edges and to search in a layer.

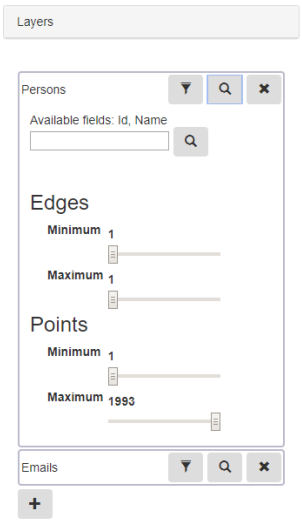

Figure 5.2: The user interface to work on layers

The implementation was also written with most of the highlighting approaches (see 4.4) optional as there often is no objectively best candidate and this allows the participants of the evaluation to find the methods that help them the most. They can be selected in this sidebar as well. There are options for visualizing node multiplicity, edge multiplicity, selection and co-occurrence (see figure 5.3).

5.1.2 Rendering

As there is no library that is flexible enough to provide the visiualizations needed while still have enough performance to render tens of thousands of nodes and edges, it has been implemented using WebGL and Three.js[1] to avoid some of the implementation overhead. This library has been chosen as it is very popular and thus is very well supported and documented.

5.1.3 Layout

Since force-directed layouts require complete physical simulation and are therefore very prone implementation errors, implementing one by hand has been avoided. This is not the focus of this thesis and would slow down progression unnecessarily. D3.js[2] is one very popular library for data visualization and it also provides a module for force-directed layouts[3]. It has been implemented using optimizations like Barnes-Hut-Approximation [BH86] and therefore can be expected to perform reasonably well.

1 https://threejs.org/
2 https://d3js.org/
3 https://github.com/d3/d3-force

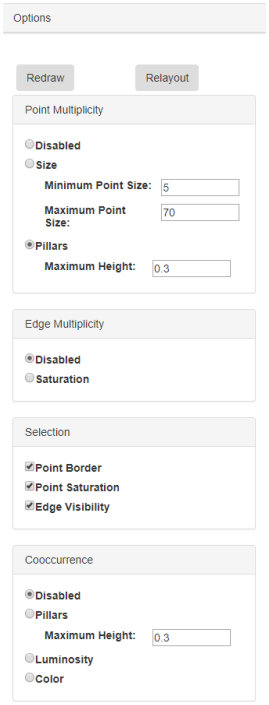

Figure 5.3: Visualization options

6 Evaluation

The implementation of this approach was evaluated by having ten subjects try to answer six questions about data visualized using this approach and afterwards rating their experience and how much they were affected by certain problems (see 8). The database used was one that has never been used during development to ensure that the results are not skewed due to having a specialized implementation for the data used. It had tables with a maximum of about 9300 entries and the visualization of the two most populated tables consisted of about 17000 nodes and 9300 edges.

The subjects were a mixture of computer science students, software engineers and one designer. They were given a short introduction how to use the user interface, but were otherwise left for themselves to explore the program, in order to evaluate the intuitiveness of the solution.

The average and median have been calculated for the results of the ratings and problems. Since the number of participants for the evaluation was even and there is no clear mid of the sorted numbers, the median is calculated as average of the numbers around the mid.

6.1 Results

6.1.1 Results of the Questions

The subjects were tasked to answer the same six questions each to a given data set using the visualization developed and implemented in this thesis. The questions can be found in 8 and the results are in table 6.1. The approach in this thesis is better suited for qualitative statements than quantitative ones, as it was mostly developed to discover trends and outliers. These quantitative questions however were chosen to be able to evaluate their answers with more expressiveness and although it is not the focus of this thesis, being able to make quantitative statements is one partial goal.

Answered correctly	49
Answered incorrectly	2
Subject could not answer	9

Table 6.1: Answers to questions (ten subjects, six questions each, so a total of 60 questions)

6.1.2 Ratings

The subjects were also tasked to rate their experience based on given questions or statements (see table 6.2) on a scale from 0 (very bad, very slow) to 4 (very good, very fast). The results are depicted in table 6.3.

Number	Text
1	How fast do you feel were you able to answer the questions?
2	How intuitive did building the graph feel?
3	How intuitive did the visualizations in the graph feel?
4	How fast did you adapt to this new visualization?
5	How fast were you able to work with the visualization after the first learning phase?
6	How intuitive did the whole process of answering the questions feel?
7	How do you rate your overall experience?
8	The pillar highlighting was helpful
9	The size highlighting was helpful

Table 6.2: Questions for the test subjects

Question \ Subjects	1	2	3	4	5	6	7	8	9	10	Median	Average
1	1	1	0	0		1	2	0	1	1	1	0,8
2	2	2	2	0	2	2	3	1	1	3	2	1,82
3	4	4	3	0	2	2	3	3	3	2	3	2,64
4	2		3	0	1	3	2	2	2	2	2	1,9
5	2	3	2	0	3	2	4	2	1	3	2	2,18
6	1	3	2	0	2	2	3	2	1	2	2	1,82
7	1	3	1	0	1	3	3	2	2	2	2	1,82
8	0	0	2	0	0	4	4	0	3	4	1	1,64
9	0	4	2	0	0	2	0	0	1	2	0,5	1,05

Table 6.3: Ratings

6.1.3 Problems

Lastly the subjects were tasked to rate how strongly they were affected by a set of problems on a scale from 0 (not at all) to 4 (very strongly). The problems can be found in table 6.4 and the results in table 6.5

Number	Text
1	I had problems to find a perspective that allowed me to see the data properly
2	Building the graph took too long
3	I did not know which layers to choose in which order
4	I just wanted to use a more traditional visualization (like a 2D graph)
5	Overlapping of nodes
6	I had problems with the implementation (too slow, bad UI, etc)
7	The given metrics did not yield much information
8	The edges did not help me answering the questions
9	The edges hindered my view to answer the questions

Table 6.4: Problems to rate

Problem \ Subject	1	2	3	4	5	6	7	8	9	10	Median	Average
1	2	2	3	4	2	2	2	2	3	2	2	2,36
2	0	4	4	4	4	4	3	4	4	4	4	3,55
3	2	2	4	4	2	2	3	3	3	3	3	2,82
4	3	1	3	4	3	0	1	2	1	1	1,5	1,86
5	2	2		4	1	2	0	0	1	4	2	1,8
6	3	3	2	4	4	2	1	3	2	3	3	2,73
7	2	1	2	4	2	2	0	1	1	1	1,5	1,59
8	3	3	1	4	1	2	0	2	1	3	2	2
9	3	0	1	4	1	2	0	1	1	2	1	1,45

Table 6.5: Ratings of the problems

7 Conclusion

In this chapter the results of this thesis will be summarized. It consists of the interpretation of the results of the evaluation (see 6) for the numerical results as well as observations of the evaluation subjects and the author. The approach will be compared to two dimensional approaches and parts of the approach, that need improvement, will be highlighted.

7.1 Results of the Evaluation

For the evaluation an implementation was used that was still a prototype, which shows in the results to some degree. However the success rate when answering questions was still rather high and apparently not many of the suggested problems were found by the subjects, but one should still keep in mind, that an unfinished version of the approach was used for this evaluation.

7.1.1 Questions

As seen in 6.1.1 49 out of the 60 questions were answered successfully and only 2 were answered incorrectly, resulting in a success rate of about 82%. This can be considered a good outcome especially when taking into account that questions asked were all more quantitative although this approach is expected to be stronger with qualitative questions. However there is obviously still much room for improvement.

It is worth noting that only 2 of the 60 questions were answered incorrectly while the subjects could not answer 9 questions. This might be a hint that the solution interactive part of the approach was a greater problem than the visualization, as the visualization was interpreted correctly, but the subjects might have failed to build it in the first place more often.

7.1.2 Ratings

According to the data shown in 6.1.2 the worst problem was the time the subjects needed to answer the questions. Only one subject did not state that the time needed was long or very long. This could have multiple reasons but according to oral statements of the subjects, the loading time when adding a new layer was just too long. This problem will be discussed further in 7.2.2.

Another interesting observation is that the subjects stated that the visualization of metrics as relative sizes (see figure 7.1) and pillars (see figure 7.2) were not helpful. Several subjects could be observed to not even use these features once instead fully relying on filters. Those, who used the pillars however, stated that they liked them a lot, but it is still a minority. One reason for this could be the quantitative nature of the questions that made quantitative visualization techniques like filter more useful in comparison to qualitative techniques like pillars. Another potential reason could be the composition of the group of subjects. They were

mostly computer science students or software engineers which often tend to use methods are require quantitative input and yield accurate quantitative result.

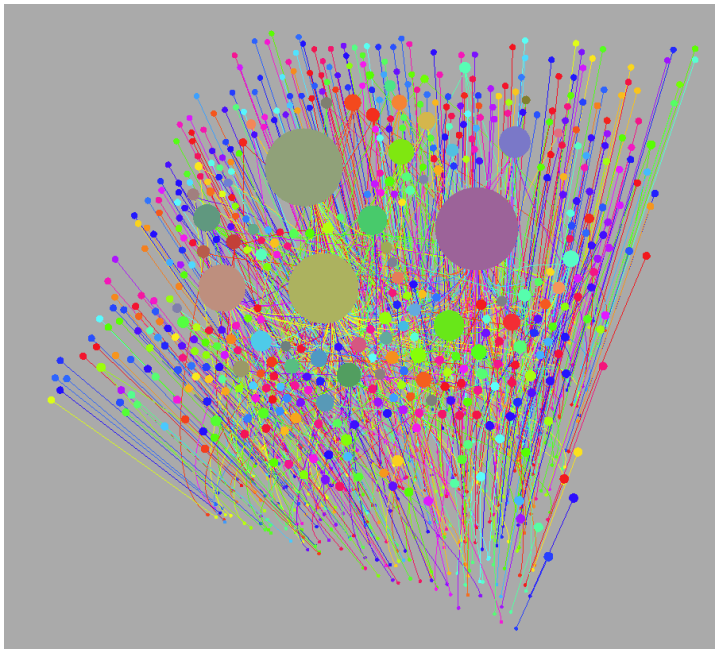

Figure 7.1: A graph visualization using node sizes to indicate multiplicities

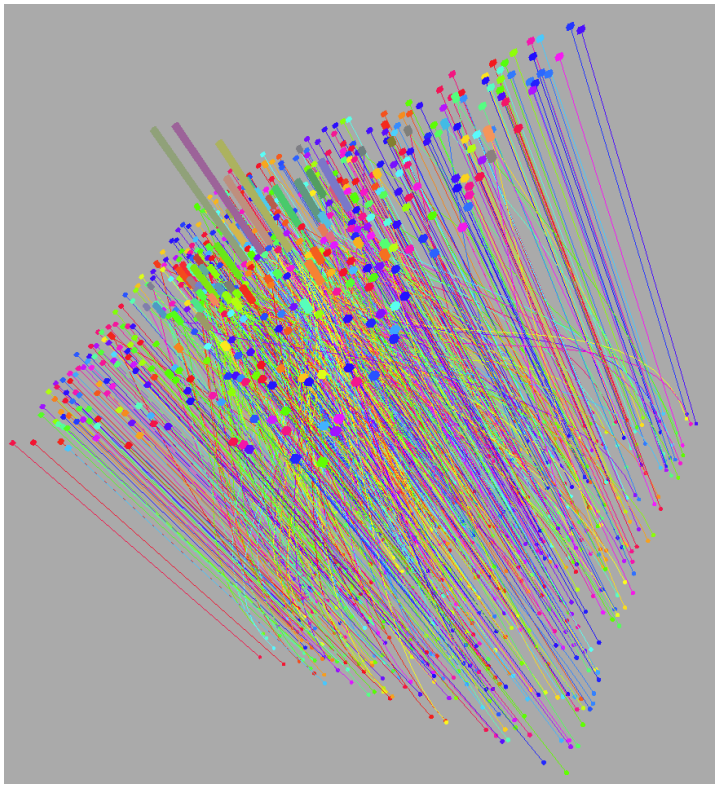

Figure 7.2: A graph visualization using pillars to indicate multiplicities

It is worth noting though that the majority of the subjects rated the visualizations as intuitive or very intuitive. This indicates that despite its problems the approach still seems to have potential.

7.1.3 Problems

As expected after the ratings before the most commonly voted problem was the time necessary to build the graph. This problem was known and has been very obvious during the evaluation, so it will discussed in more detail in 7.2.2.

Although the ratings of the subjects were not that distinctive it is worth noting that many subjects stated that the given metrics did not yield much information. This might be especially interesting in combination with the rating before stating that the size and pillar visualizations were not very helpful, since this could either mean that the metrics were acceptable, but their visualizations were insufficient or the metrics were actually the problem. This question can not be answered in this thesis.

The majority of participants of the evaluation stated that they would have preferred a more traditional visualization, but only one stated that they felt this strongly.

According to the subjects of the evaluation the edges were not a problem. Considering of the size of the used database, this can be considered a strong indicator that the solution already scales pretty well despite not having a technique like edge bundling included.

7.1.4 Surprises

It is positively surprising to see such a high success rate of answering questions that are not the strongest focus of this thesis about a database of such extent that was unknown at the time of implementation. This is a strong indicator that the goal of a generic approach was met.

It is also surprising to see that the implementation was not considered a bigger problem considering how strong the opinion about the speed of the implementation was. This however should have no direct affect on the evaluation of the approach itself.

7.2 Problems of the Approach

The approach and implementation have some general and significant problems that will be discussed here.

7.2.1 Edges

The edges of the graph have shown to still be more of a drawback than being helpful. When displaying few edges, they are probably about as useful in a two dimensional visualization, but when their amount is increased, for instance to several thousand between a layer, then the viewer is unable to gain any information from them (see figure 7.3). According to the evaluation the approach mitigated the problem well, so the edges at least do not obstruct the view to the relevant nodes, but they are still not useful on their own.

This changes when nodes are selected and only connected edges are visualized, as the amount of edges often shrinks enough to be expressive again.

In summary the layered graph and visualization techniques like pillars seem to help with the problem of having a plethora of edges, but this problem is still unsolved.

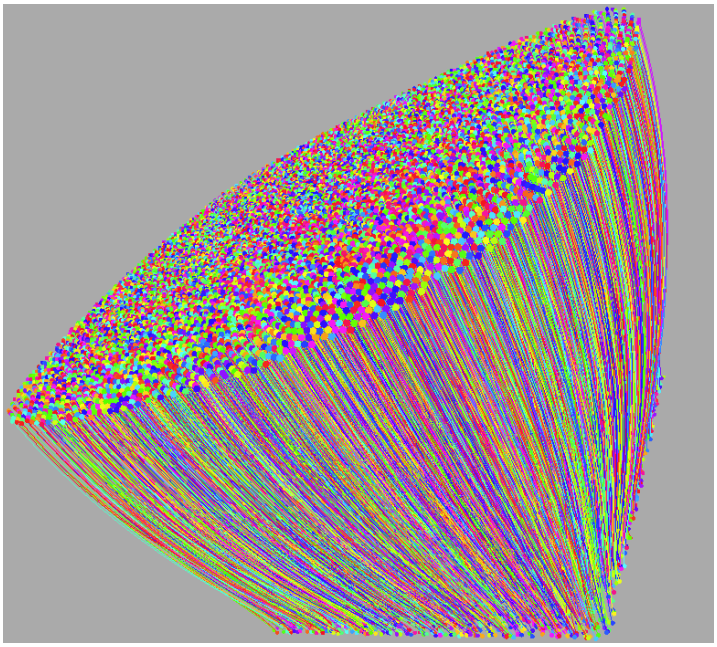

Figure 7.3: A graph consisting of about 8000 nodes in the first layer, 9300 nodes in the top layer and 9300 edges in between.

7.2.2 Performance

The biggest problem of the implementation has shown to be the performance. The bottleneck here is the force-directed layout, which is implemented using D3.js. It can take even a minute to finish a layout for about 15000 nodes, which is obviously unacceptable and has been highly criticized during the evaluation. This problem prevents the user from 'just testing things', which is necessary to properly understand the given data. At the current state the prototype can be considered not really usable due to this.

The underlying problem here is probably that the module of D3.js for force-directed layout is designed for a problem size one or several magnitudes smaller than this approach. A faster algorithm for force-directed layout (for instance like developed by Toosi and Nikolov [TN16]) might solve this problem or a completely different approach to compute a layout has to be chosen.

7.2.3 Layout

Choosing the correct parameters for the forces in a force-directed layout is always a problem. This implementation is no exception and the problem still remains. The intention of using this layout was to naturally form clusters of nodes that all together have the same direct predecessor and this mostly works, but there are always outliers that are not properly pulled to the cluster. Increasing the forces of the edge contraction however results in different worse problems like overlapping nodes.

7.2.4 Usefulness of Multiple Layers

In theory an unlimited amount of layers can be stacked using this approach. However since most oft the visualization options only apply to the topmost layer and the edges tend to hide layers between the topmost and the first layer, they have shown to not yield any information. In addition to that no database could be publicly found that would proper use of having more than two layers at once visualized. This suggest that specializing this approach to only support two layers might be a good idea.

7.2.5 Order of Layers

It has been stated multiple times by the participants of the evaluation that selecting the layers in the correct order is not always intuitive. This in combination of the slow process of adding the layers is a big issue. The participants asked for a quick option to simply switch the order of the current layers to correct this, which would support the idea of specializing to only visualizing a maximum of two layers at once. But still this would require a significant boost in performance since the layout would need to be re-calculated whenever the layers would be switched.

7.3 Comparison to 2D

The obvious question that arises when creating a three dimensional graph visualization is for the benefits over a two dimensional visualization. Two dimensional visualizations have been researched extensively so adding a new visualization needs to be justified. The advantages and disadvantages of the approach developed in this thesis will be discussed here.

7.3.1 Advantages

Although this approach still has its problems with the visualization of edges (see 7.2.1), the evaluation has shown that it does scale very well. Since the scaling with an increasing amount of edges is one of the biggest drawbacks of two dimensional node-link visualizations, this is a clear benefit. As already stated there might even be potential to further improve this scaling.

The extend at which two dimensional node-link visualizations have been researched also improves this approach as it allows to use visualization techniques and layouts that are originally meant for two dimensional visualizations. However in addition to this this approach can also use techniques like three dimensional pillars. This flexibility is also one big advantage of this approach.

7.3.2 Disadvantages

Two dimensional do obviously still have advantages over this approach though. The visualization is very dependent on the point of view, which makes it near impossible to view it statically. Printing it on a piece of paper would make very little sense.

In addition to relying on the user to search for a good perspective, it also requires interaction to build the graph in the first place. This again makes it unusable for many use cases.

Lastly this approach is still far from perfect and suffers from problems like lack of intuitiveness. It has clearly been stated that finding the correct order to visualize certain aspects of the graph is still not intuitive and a problem. Two dimensional visualizations have been used to centuries and viewers are very familiar with them.

7.4 Future Work

This approach can be considered a step in the right direction, but it still has some flaws. Some ideas for solutions will be listed here.

7.4.1 Layout

The layout has shown to be a problem for multiple reasons. It is too slow (see 7.2.2) and finding the right parameters is hard and might even be impossible for the generic case (see 7.2.3). It is worth considering a completely different approach for finding a good layout. At the very least an algorithm for force-directed layouts should be used that is designed to scale into the ten thousands, potentially hundred thousands, as this was the original goal of this approach.

7.4.2 Edges

Originally the intention of this thesis was to use edge bundling to make visualizing edges not only a drawback that needs to be minimized but actually a help to understand the data. It was planned to enable the viewer to see trends in the data and make quantitative statements about it.

This idea has not been included in this thesis since there it was aimed at generic data and automatically finding useful edge attributes to bundle them is currently not possible (to the knowledge of the author at least). One potential solution to this problem would be to also offer interactive edge bundling. Letting the user choose the criteria to bundle by, would solve this problem, but the acceptance with the users would have to be tested as well.

7.4.3 Layers

Specializing the approach to only visualize a maximum of two layers might yield great benefits, as it would improve usability and allow more specialized options like switching the layers, which has been requested by the participants of the evaluation. This step might also help with the layout, as a potentially more specific and faster solution could be found.

Bibliography

[AHH17] James Abello, Fred Hohman, and Duen Horng Chau. 3D Exploration of Graph Layers via Vertex Cloning. pages 3–4, 2017.

[BC05] Therese Biedl and Timothy M. Chan. A note on 3D orthogonal graph drawing. *Discrete Applied Mathematics*, 148(2):189–193, 2005.

[BDS04] Ulrik Brandes, Tim Dwyer, and Falk Schreiber. Visualizing Related Metabolic Pathways in Two and a Half Dimensions. *Graph Drawing*, 2912:111–122, 2004.

[BH86] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324:446–449, 12 1986.

[BVD19] Wolfgang Buschel, Stefan Vogt, and Raimund Dachselt. Augmented Reality Graph Visualizations: Investigation of Visual Styles in 3D Node-Link Diagrams. *IEEE Computer Graphics and Applications*, 1716(c):1–1, 2019.

[DCW+18] Adam Drogemuller, Andrew Cunningham, James Walsh, Maxime Cordeil, William Ross, and Bruce Thomas. Evaluating Navigation Techniques for 3D Graph Visualizations in Virtual Reality. *2018 International Symposium on Big Data Visual and Immersive Analytics, BDVA 2018*, pages 1–10, 2018.

[HN05] Seok Hee Hong and Nikola S. Nikolov. Layered drawings of directed graphs in three dimensions. *Conferences in Research and Practice in Information Technology Series*, 45:69–74, 2005.

[HNT07] Seok-hee Hong, Nikola S Nikolov, and Alexandre Tarassov. A 2 . 5D Hierarchical Drawing of Directed Graphs. *Most*, 11(2):371–396, 2007.

[Hol06] Danny Holten. Hierchical Edge Bundles: Visualizaiton of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[Hu06] Yifan Hu. The Mathematica ® Journal Efficient, High-Quality Force-Directed Graph Drawing. *Mathematica Journal*, 10:37–71, 01 2006.

[KB67] AN Kolmogorov and Ya M Barzdin. On the realization of nets in 3-dimensional space. *Problems in Cybernetics*, 8(261-268):259–260, 1967.

[LBA10] Antoine Lambert, Romain Bourqui, and David Auber. 3D Edge Bundling for Geographical Data Visualization. In *2010 14th International Conference Information Visualisation*, pages 329–335. IEEE, jul 2010.

[SFP+18] Johanna Schmidt, Dominik Fleischmann, Bernhard Preim, Norbert Brandle, and Gabriel Mistelbauer. Popup-Plots: Warping Temporal Data Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2626(10):1–15, 2018.

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man and Cybernetics*, 1981.

[TN16] F. G. Toosi and N. S. Nikolov. Vertex-neighboring multilevel force-directed graph drawing. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 002996–003001, Oct 2016.

[Tut63] W. T. Tutte. How to Draw a Graph. *Proceedings of the London Mathematical Society*, s3-13(1):743–767, 1963.

8 Appendix A: Evaluation Form

Evalution

March 2019

This evalution consists of questions that your are supposed to answer using the given graph visualization as well as some tables to grade your experience. You will be given the visualization of a database containing information about the emails used in the lawsuit against Hillary Clinton regarding the controversy over the use of personal email accounts on non-government servers during her time as the United States Secretary of State[1]. This data has not been altered, so keep in mind, that the table "Persons" in connection to "Emails" are their senders.

1 Explanation of the UI

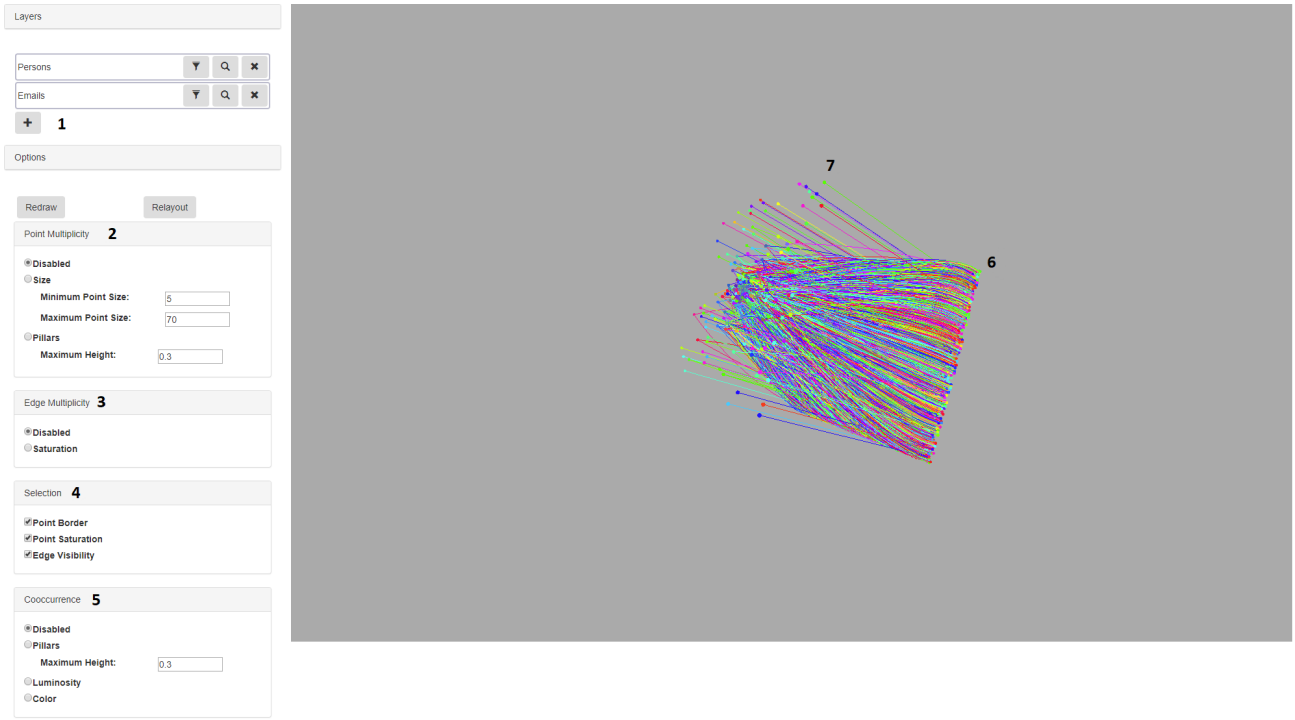

Figure 1: The UI of the graph visualization

1. Here you can add and remove layers of the graph. Their layout and properties like multiplicity depend on their previous layer, so order matters.
2. Here you can choose how point multiplicity is visualized. Point multiplicity is the number of connections between a point and points in the layer below.
3. Here you can choose how edge multiplcity is visualized. Edge multiplicity is the number of connections an edge represents.

[1] Source https://www.kaggle.com/kaggle/hillary-clinton-emails, at 26.3.2019

4. Here you can choose how selection of nodes is visualized.
5. Here you can choose how co-occurrence is visualized. The co-occurrence of a point is always in regard to a selected point (so selection is necessary) and indicates how often using how many connections it is connected via the previous layer. That means more simplified how often these points appear together.
6. This is the bottom layer of the graph. You can see what it represents in the layer panel at the top left.
7. This is the top layer of the graph. You can see what it represents in the layer panel at the top left.

2 Tasks

Please try to answer the following question using the given graph visualization, but only select one answer per question.

1. Who was the most common recipient of emails?

   ☐ _____
   ☐ I can see that this question cannot be answered by the given data.
   ☐ I was unable to fully answer this question.

2. Which alias was used the most in the emails?

   ☐ _____
   ☐ I can see that this question cannot be answered by the given data.
   ☐ I was unable to fully answer this question.

3. Who has the most aliases?

   ☐ _____
   ☐ I can see that this question cannot be answered by the given data.
   ☐ I was unable to fully answer this question.

4. Find the email with the most recipients. Is there an email that it shares the most recipients with? What is its id?

   ☐ Yes there is one:_____
   ☐ There is none.
   ☐ I can see that this question cannot be answered by the given data.
   ☐ I was unable to fully answer this question.

5. Are there shared aliases?

   ☐ Yes.
   ☐ No.
   ☐ I can see that this question cannot be answered by the given data.
   ☐ I was unable to fully answer this question.

6. How many emails did Chelsea Clinton write?

☐ _____
☐ I can see that this question cannot be answered by the given data.
☐ I was unable to fully answer this question.

3 Grading

3.1 Rating

In this section you should grade several aspects of your experience answering the questions above. The rating spreads from 0 (very bad, very slow) to 4 (very good, very fast). Please check only one box per line.

Keep in mind that this grading is supposed to be subjective, so please do not use objective measurements like checking how many minutes you actually took to answer a question.

	0	**1**	**2**	**3**	**4**
How fast do you feel were you able to answer the questions?	☐	☐	☐	☐	☐
How intuitive did building the graph feel?	☐	☐	☐	☐	☐
How intuitive did the visualizations in the graph feel?	☐	☐	☐	☐	☐
How fast did you adapt to this new visualization?	☐	☐	☐	☐	☐
How fast were you able to work with the visualization after the first learning phase?	☐	☐	☐	☐	☐
How intuitive did the whole process of answering the questions feel?	☐	☐	☐	☐	☐
How do you rate your overall experience?	☐	☐	☐	☐	☐
The pillar highlighting was helpful	☐	☐	☐	☐	☐
The size highlighting was helpful	☐	☐	☐	☐	☐

3.2 Problems

Please grade now how strongly you were affected by certain problems. The rating spreads again from 0 (not at all) to 4 (very strongly). Please check only one box per line.

	0	**1**	**2**	**3**	**4**
I had problems to find a perspective that allowed me to see the data properly	☐	☐	☐	☐	☐
Building the graph took too long	☐	☐	☐	☐	☐
I did not know which layers to choose in which order	☐	☐	☐	☐	☐
I just wanted to use a more traditional visualization (like a 2D graph)	☐	☐	☐	☐	☐
Overlapping of nodes	☐	☐	☐	☐	☐
I had problems with the implementation (too slow, bad UI, etc)	☐	☐	☐	☐	☐
The given metrics did not yield much information	☐	☐	☐	☐	☐
The edges did not help me answering the questions	☐	☐	☐	☐	☐
The edges hindered my view to answer the questions	☐	☐	☐	☐	☐