Supporting Graph Editing in Visual Representations

Dissertation
zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Stefan Gladisch
geboren am 11.06.1987 in Rostock
wohnhaft in Rostock

Rostock, 29.01.2016

urn:nbn:de:gbv:28-diss2016-0079-0

Betreuer

▷ *Prof. Dr.-Ing. habil. Heidrun Schumann*
Universität Rostock, Deutschland

Externe Gutachter

▷ *Prof. Dr.-Ing. Raimund Dachselt*
Technische Universität Dresden, Deutschland

▷ *Prof. Dr.-Ing. Andreas Kerren*
Linnaeus University, Schweden

Datum der Einreichung

▷ *29.01.2016*

Datum der Verteidigung

▷ *01.07.2016*

Schlagwörter

Editieren von Graphen, Graphvisualisierung, Linsen.

Keywords

Graph editing, graph visualization, lenses.

Klassifikation / Classification (ACM CCS 2012)

Human-centered computing → Information visualization, Human-centered computing → Visualization theory, concepts and paradigms.

Kurzfassung

Visualisierungsverfahren sind das etablierte Mittel, um Nutzer bei der Extraktion nutzvoller Informationen aus Graphen zu unterstützen. Obwohl solche Informationen die Notwendigkeit des Editierens eines Graphen aufzeigen können und auch Aufschluss über den Effekt zuvor durchgeführter Datenänderungen vermitteln können, werden Graphen meist in externen, alphanumerischen Datenrepräsentationen editiert. Diese Separierung erschwert die Arbeit des Nutzers.

Das Ziel der vorliegenden Dissertation ist es, Lösungen zur Unterstützung des direkten Editierens von Graphen in visuellen Repräsentationen zur Analyse von Graphen bereitzustellen. Dafür wird zunächst eine konzeptuelle Sicht auf die Aufgaben des Nutzers entwickelt. Auf dieser Basis werden anschließend mehrere neue Verfahren eingeführt, welche das "visuelle editieren" der verschiedenen Datenaspekte von Graphen – der Graphstruktur sowie dazu assoziierte Attributwerte – ermöglichen. Dabei werden verschiedene, visuelle Graphrepräsentationen berücksichtigt, welche die Daten in geeigneter Form kommunizieren können.

Das Arbeiten mit visuellen Graphrepäsentationen beinhaltet meist das Navigieren zu verschiedenen Teilmengen der Daten sowie das Anpassen der Repräsentation, um bestimmte Details zu betrachten. Um den Nutzer in dieser Hinsicht besser zu unterstützen, wurde zusätzlich ein Verfahren zur Bereitstellung von Navigationsempfehlungen zu interessanten Daten entwickelt und das Konzept von Linsen betrachtet.

Die Anwendbarkeit und Eignung der in dieser Dissertation entwickelten Verfahren wird an Hand verschiedener Anwendungsfälle demonstriert.

Abstract

Visualization approaches are the established means to support the user in extracting useful information from graphs. Although such information can reveal the need for editing a graph and can provide insights about the effect of previously applied data changes, graph editing is still mostly carried out in external, alphanumeric representations. This separate approach complicates the user's work.

The goal of this thesis is to provide solutions for supporting the direct editing of graphs in visual representations for analyzing graphs. For that, a conceptual view on the user's tasks is established first. On this basis, several novel approaches to "visually edit" the different data aspects of graphs – the graph's structure and associated attribute values – are introduced. Thereby, different visual graph representations suitable for communicating the data are considered.

Usually, working with visual graph representations includes navigating to different subsets of the data and adjusting the representation for viewing certain details. To better support the user in this respect, an approach for providing navigation recommendations to interesting data is presented additionally and the concept of lenses is investigated.

The applicability and suitability of the approaches developed within this thesis is demonstrated with several use cases.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Anfertigung dieser Dissertation unterstützt haben. Zu aller erst danke ich Professorin Heidrun Schumann für Ihre stets konstruktive und zielführende Kritik sowie positive Sicht auf die Dinge, welche für mich sehr motivierend war. Ebenso danke ich Ihr, Professor Raimund Dachselt und Professor Andreas Kerren für die bereitwillige Übername der Begutachtung dieser Arbeit. Ganz besonders danke ich ebenfalls Dr.-Ing. habil. Christian Tominski der mich über den gesamten Zeitraum an der Universität Rostock unterstützt hat.

Darüber hinaus danke ich Dr.-Ing. Martin Luboschik, Christian Eichner, Dr.-Ing. Steffen Hadlak, Dr.-Ing. Hans-Jörg Schulz, Falko Löffler, Dr.-Ing. Axel Radloff, Steve Dübel, Martin Nyolt, Martin Röhlig und Ulrike Kister für die fachlichen Diskusionen sowie inspirierenden Ideen. Weiter Dank gebührt Valerius Weigandt und Arne Neuber die mich als wissenschaftliche Hilfskräfte bei der Implementierung zweier neu entwickelter Verfahren unterstützt haben.

Nicht zuletzt danke ich meinen Eltern, die mir das Studium und die anschließende Promotion überhaupt erst ermöglicht haben. Speziell möchte ich mich auch bei meiner Partnerin Anne bedanken, die mir stets viel Verständnis für längere Arbeitstage entgegengebracht hat und mich abgelenkt hat, wenn es notwendig war.

Contents

1. Introduction **1**

2. Basics and Related Work **7**
2.1. Terms and Definitions . . . 8
2.2. Graph Visualization . . . 9
2.2.1. Data-State-Reference Model . . . 9
2.2.2. Visualization of Structure . . . 10
2.2.3. Visualization of Attributes . . . 15
2.3. Graph Editing . . . 23
2.3.1. Graph Editing Using Tabular Representations . . . 23
2.3.2. Graph Editing Using Node-Link Representations . . . 24
2.3.3. General Editing Model . . . 26
2.4. Summary, Unsolved Problems and Goals . . . 28

3. A Conceptual View on Tasks of Visual Graph Editing **31**
3.1. Tasks of Visual Graph Editing . . . 31
3.1.1. *Select or Specify Data to Edit* . . . 32
3.1.2. *Apply Edit Operation* . . . 33
3.1.3. *Analyze Data* . . . 34
3.1.4. *Adjust Representation* . . . 36
3.2. A Task-Transition Model for Visual Graph Editing . . . 38
3.3. Summary . . . 43

4. Novel Techniques for Editing the Graph's Structure **45**
4.1. Semi-Automatic Editing of Graphs with Customized Layouts . . . 47
4.1.1. Problem Analysis . . . 47
4.1.2. Approach . . . 48
4.1.3. Implementation . . . 59
4.1.4. Use Case and Evaluation . . . 59
4.1.5. Discussion . . . 66
4.2. Using Matrix Representations for Graph Editing . . . 67
4.2.1. Problem Analysis . . . 67
4.2.2. Approach . . . 69
4.2.3. Implementation . . . 72
4.2.4. Use Case . . . 72
4.2.5. Discussion . . . 73

4.3. Summary 74

5. Novel Techniques for Editing the Graph's Attribute Values 77
5.1. Editing Node Attribute Values by Moving Nodes in the Graph Layout . . 81
5.1.1. Problem Analysis 81
5.1.2. Approach 82
5.1.3. Implementation 92
5.1.4. Use Case and Evaluation 92
5.1.5. Discussion 96
5.2. Integrating Radial Controls for Editing Graph Attribute Values 97
5.2.1. Problem Analysis 98
5.2.2. Approach 99
5.2.3. Implementation 107
5.2.4. Use Case and Evaluation 108
5.2.5. Discussion 111
5.3. Summary 112

6. User Support for Visual Graph Editing 115
6.1. Navigation Recommendations for Exploring Hierarchical Graphs 117
6.1.1. Problem Analysis 117
6.1.2. Approach 120
6.1.3. Implementation 126
6.1.4. Use Case 127
6.1.5. Discussion 128
6.2. A Comprehensive Overview of Lenses 129
6.2.1. Problem Analysis 130
6.2.2. Definition of Interactive Lenses in Visualization 130
6.2.3. Categorization of Lenses according to Data and Tasks 138
6.2.4. Lens Interaction and Display 147
6.2.5. A Demonstrating Example: the *EditLens* 149
6.2.6. Discussion 151
6.3. Summary 152

7. Summary and Outlook 153

8. Appendix: An Orthogonal Edge Routing Algorithm for the *EditLens* 159

Bibliography 167

List of Scientific Publications and Lectures 189

Thesis Statements 193

List of Figures

2.1. The *data-state-reference model* 9
2.2. Exemplary layouts of node-link representations 12
2.3. Different orderings of a matrix representation 13
2.4. Exemplary implicit representations 14
2.5. Exemplary hybrid representations 15
2.6. Exemplary representations with attribute-dependent layout 17
2.7. Exemplary node-link representations with attribute-independent layout . . 19
2.8. Exemplary matrix representations with attribute-independent layout . . . 20
2.9. Exemplary implicit representations with attribute-independent 21
2.10. Exemplary tabular representation and coordinated views 22
2.11. Graph editing using an interlinked table 24
2.12. Exemplary techniques for graph editing in node-link representations . . . 25
2.13. Visual graph editing from the perspective of the *data-state-reference model* 27
2.14. The *general editing model* 28

3.1. The *extended general editing model* 39
3.2. Illustration of multiple selections 41

4.1. Schematic depiction of tasks supported by the *EditLens* 50
4.2. Determining empty space within the *EditLens* 52
4.3. Illustration of different node placement strategies 54
4.4. Performance test results of an orthogonal edge routing algorithm 56
4.5. Touch gestures used in the *EditLens* prototype. 58
4.6. The *PluriNetWork* 60
4.7. Modeling the editing procedure of the *PluriNetwork* 62
4.8. The *EditLens* applied to the *PluriNetwork* 63
4.9. Visualization of the *PluriNetWork* with the *EditLens* prototype 65
4.10. Direct insertion of an edge in a matrix representation 70
4.11. Direct insertion of multiple edges in a matrix representation 71
4.12. Edge editing in an interlinked view setting 72
4.13. Visual editing of edges of a co-occurrence network 73

5.1. Modeling the common attribute value editing procedure 78
5.2. Node-link representation with scatterplot layout 83
5.3. Cluster representative and fan out. 84
5.4. Editing a quantitative node attribute value in a scatterplot layout 87

5.5. Editing two quantitative node attribute values simultaneously 88
5.6. Node-link representation with semantic substrates layout 89
5.7. Editing a qualitative node attribute value in a semantic substrates layout 90
5.8. Reducing editing costs with an off-screen visualization 91
5.9. Visual editing of attribute data of a wireless network 93
5.10. Visual editing of attribute data of a document network 95
5.11. Node-link representation with attribute-independent layout 100
5.12. Basic features of radial menus and radial sliders 102
5.13. Step-by-step editing of attribute values using an editing interface 105
5.14. Radial menus and radial sliders with integrated visual cues 106
5.15. Visual editing of attribute data of a social network 109

6.1. Navigation in a hierarchical graph 118
6.2. The problem of navigation 119
6.3. Different definitions of focus and context 121
6.4. Techniques for recommending navigation to off-screen nodes 124
6.5. Expand animation as vertical navigation recommendation 125
6.6. Implementation of different kinds of navigation recommendations 127
6.7. Schematic depiction of an interactive lens. 130
6.8. Lenses with different shapes 131
6.9. Lenses with different orientation 133
6.10. Conceptual model of lenses in visualization 134
6.11. Basic lens functions 135
6.12. Lens effects beyond the confines of the lens interior 136
6.13. Lenses for different data types 139
6.14. Lenses for different tasks 144
6.15. Examples of interaction modalities and display settings used for lenses . . 148
6.16. Repositioning the *EditLens* using touch interaction 150

8.1. Intersection test of orthogonal lines 160
8.2. Illustration of lanes between nodes used for edge routes 162
8.3. Two results achieved by the new edge routing algorithm 164
8.4. Comparison of edge routing runtime performance 165

List of Tables

6.1. Categorization of lenses according to data types and tasks 146

1. Introduction

Motivation

Today, we are facing huge amounts of digital data and analyzing them is more important than ever. According to the results of a study from *EMC Corporation* in 2014, the amount of digital data produced by mankind will multiply 10-fold between 2013 and 2020 from 4.4 trillion gigabytes to 44 trillion gigabytes. Although the amount of data that is considered as useful for analysis is estimated by 22% in 2013, less than 5% could actually be analyzed – leaving a massive amount of data lost in the digital universe [EMC15]. When analyzing datasets, it is not only the data objects that are of interest, but also the relations between them. Common examples are relationships between people in social networks, traffic between hosts in computer networks, relations between packages or classes in software systems and interactions between proteins or species in biological systems [KPW14; Tam07]. Such relations between data objects can be represented with graphs, where the graph's nodes depict data objects and the graph's edges depict relations. To support the understanding and the extraction of useful information from data represented as graphs, visualization approaches transforming graphs into visual representations have been established. The motivation for these approaches is the human perception that is mostly dedicated to processing visual information.

Besides the need for analyzing graphs, there is also the need for editing graphs due to various reasons. Examples are graphs containing missing nodes or edges that need to be inserted, outdated graphs that need to be updated, graphs that need to be edited in context of what-if scenarios, or graphs that are newly created, for example in software modeling with UML. Since editing often incorporates human understanding and judgment, interactive visualization approaches are also helpful in this regard. They can support the user in understanding the graph they are working with, in identifying data subsets relevant for editing and in comprehending changes to the graph caused by editing.

Whereas several powerful systems like *CGV* [TAS09], *Gephi* [BHJ09], *Tulip* [Aub04] or *Cytoscape* [Sha+03] exist offering graph visualization and interactive means to facilitate the visual analysis of graphs, there is only limited support in these systems to edit graphs. Commonly, graph editing is carried out by using text editors or diagramming tools like *Microsoft Visio*, *Visual Paradigm* or *Enterprise Architect*. These systems however do not support the visual analysis of graphs to a full extent.

Problem Description and Goals

In various situations, graph analysis and editing are interrelated. On the one hand, a meaningful analysis might incorporate the correction of erroneous nodes or edges beforehand [Kan+11]. On the other hand, the need for editing a graph can be revealed by analyzing the data. Moreover, after editing a graph, it is usually necessary to verify the editing outcome by analyzing the data. In such situations, users are forced to work with different systems side by side to accomplish their tasks properly. This separative approach implies disadvantages. Users have to switch between different systems, which produces additional cognitive load. Due to different data representations across systems, users have to retain a considerable amount of information in short term memory. This includes information about how to relocate data objects for instance. Furthermore, users have to keep in mind how to work with each system, as interaction might differ across systems. All this may complicate the user's work.

The reason for separate graph analysis systems and graph editing systems may be that they address different overarching goals: Generally, Graph analysis has the goal to set up or test hypotheses about the data, whereas graph editing aims to change the data. However, the user's tasks to achieve these goals are partially similar and thus supported by visualization and interaction techniques in both kinds of systems. Examples include identifying and selecting nodes and edges of interest for obtaining their details or for editing them. Support for these tasks is provided by node-link representations, pan and zoom navigation, as well as interactive selection techniques.

The interrelations between visual graph analysis and graph editing motivate the development of systems supporting both to a full extent. One way to achieve this is to extend graph analysis systems to additionally support all tasks involved in graph editing. Since graph analysis systems offer various visual representations with individual strengths and weaknesses for supporting specific analysis tasks, there is a need to investigate how to use these representations to additionally support *visual* graph editing tasks. This comprises support for both, tasks for editing the graph's structure (i.e., structural graph editing) and tasks for editing attribute data associated with graphs. Since current techniques only facilitate structural editing of graphs with moderate size in node-link representations, there is a need to support further tasks and alternative representations. This, however, is difficult as a conceptual view on tasks of visual graph editing is missing.

Thus, the goal of this thesis is to provide a conceptual view on tasks of visual graph editing and to develop novel interactive techniques for their support. Thereby, a focus is set on supporting tasks for editing the graph's structure and the graph's associated attribute values.

Several challenges arise along with these goals. First, the user's tasks have to be described. Because there is no previous work in literature to build on and graph editing involves many different aspects besides changing the data, this is challenging. From this set of tasks, those must be identified which are already supported across existing systems separated from those which still need to be supported. In order to develop adequate techniques for interactively accomplishing the latter tasks, it is necessary to examine the interplay of all tasks involved in visual graph editing. The development of such techniques poses additional challenges as it also incorporates the design of appropriate interactions and strategies for visual feedback. Moreover, as graph editing may significantly influence the layout of the representation, another challenge is to preserve the user's mental map and to consider quality criteria and layout constraints during editing.

Approach

The main approach of this thesis can be summarized in four steps:

1. **Fundamental categories for tasks of visual graph editing are described and their current support is examined:** On the basis of existing work, fundamental categories of tasks of visual graph editing are set up and described in detail. The support for these tasks by existing visualization and interaction techniques is reviewed afterwards. This way, tasks which are currently not or poorly supported are identified.

2. **The interplay of tasks of visual graph editing is modeled:** As already indicated, graph analysis and editing tasks overlap, but their interplay has not been modeled before. Thus, a model describing the relation and processing order of all tasks involved in visual graph editing is developed.

3. **Novel approaches for visual graph editing are developed:** Novel approaches supporting tasks for editing the graph's structure and the graph's attribute values are developed. This is exemplarily done for two kinds of representations visualizing the graph's structure (node-link and matrix representations) and two kinds of representations visualizing attribute data associated with graphs (representations with attribute-dependent layout and representations with attribute-independent layout). Thereby, tasks which are currently not or only poorly supported are focused.

4. **User support for graph editing is investigated:** As described in [And+10], investigating how to support the user is crucial but challenging. A major issue, which requires special user support, is the fact, that visual representations of graphs usually cannot encode all the data aspects at once. This means, some data aspects important for accomplishing the user's tasks might not be visible. To alleviate this issue, user support with navigation recommendations to interesting data and interactive lenses are investigated.

Outline and Contribution

This thesis is divided into seven chapters which are summarized in the following.

Chapter 2: This chapter provides important basics of this thesis. Definitions of essential terms are provided, graph visualization and graph editing are motivated and described, and existing approaches in literature are reviewed. This work is partly based on the technical report *"Mapping Tasks to Interactions for Graph Exploration and Graph Editing on Interactive Surfaces"* [Gla+15c] published on *arxiv.org* in 2015 and the poster *"Mapping Tasks to Interactions for Graph Exploration and Graph Editing"* [Gla+15b], which was presented at the *InfoVis* conference 2015. At the end of this chapter, unsolved problems, goals and challenges are discussed.

Chapter 3: This chapter contributes a conceptual view on tasks involved in visual graph editing. Fundamental categories for tasks of visual graph editing are described, their interplay is modeled and their current support with existing visualization and interaction techniques is examined.

Chapter 4: This chapter focuses on structural graph editing. Two novel approaches supporting tasks for editing the graph's structure in node-link and matrix representations are introduced. The first approach addresses region-oriented editing in node-link representations with established and customized layouts and was motivated by an application example from bio-informatics. It has been presented at the *EuroVis* conference 2014 and was published in the journal *Computer Graphics Forum* 2014 under the title *"Semiautomatic Editing of Graphs with customized Layouts"* [Gla+14]. The second approach addresses the editing of edges in matrix representations and aims to alleviate high interaction costs associated with traditional techniques for visually editing edges. It has been presented as the poster *"Toward Using Matrix Visualizations for Graph Editing"* [Gla+15a] at the *InfoVis* conference 2015.

Chapter 5: This chapter addresses the editing of the graph's attribute values. Two novel approaches are introduced based on representations with attribute-dependent and attribute-independent layouts. Graph attributes typically have qualitative or quantitative scales. This distinction needs to be considered when editing their values. Hence, approaches addressing both are provided. The first approach addresses node-link representations with an attribute-dependent layout and enables users to edit node attribute values directly. This approach was submitted to *EuroVis* conference 2016 with the title *"Visual Editing of Node Attribute Data of Multivariate Graphs"*. The second approach enables users to edit node and edge attribute values through an editing interface integrated into node-link representations with an attribute-independent layout. This approach was presented as the poster *"Toward Integrated Exploration and Manipulation of Data Attributes in Graphs"* [GT14] at the *InfoVis* conference 2014.

Chapter 6: This chapter addresses the user support in situations where the graph representation cannot encode all the data at once. For this purpose, a novel approach is introduced that facilitates navigation to potentially interesting data. This approach has been presented at the *ISVC* conference in 2013 and appeared in its proceedings under the title *"Navigation Recommendations for Exploring Hierarchical Graphs"* [GST13]. Moreover, the support for analyzing and editing certain details in local regions of interest is addressed with a comprehensive overview of lenses in visualization. This overview is based on the state-of-the-art report *"A Survey on Interactive Lenses in Visualization"* which was presented at the *EuroVis* conference 2014 [Tom+14]. An extended version of this report with the title *"Interactive Lenses in Visualization: An Extended Survey"* was submitted to the journal *Computer Graphics Forum*.

Chapter 7 : The thesis concludes with a summary and a description of unsolved problems as well as an outlook for future work.

2. Basics and Related Work

This chapter provides the basics concerning graphs, graph visualization and graph editing and is divided into three major sections respectively. The first Section 2.1 provides a formal definition of graphs according to their structure and attributes. Based on this, Section 2.2 describes the visualization of graphs and includes a review of existing approaches. Section 2.3 addresses the editing of graphs similarly. Finally, Section 2.4 provides a summary, discusses open problems and challenges for graph editing and describes goals of this thesis. Existing work concerning tasks for graph editing and analysis is addressed in Chapter 3. A review concerning interactive lenses is provided in Chapter 6.

Contents

2.1. Terms and Definitions 8
2.2. Graph Visualization 9
2.2.1. Data-State-Reference Model 9
2.2.2. Visualization of Structure 10
2.2.3. Visualization of Attributes 15
2.3. Graph Editing 23
2.3.1. Graph Editing Using Tabular Representations 23
2.3.2. Graph Editing Using Node-Link Representations 24
2.3.3. General Editing Model 26
2.4. Summary, Unsolved Problems and Goals 28

The descriptions of graph visualization and graph editing systems included in this chapter are based on the descriptions of the following two publications I contributed to as the first author:

[Gla+15b] S. Gladisch, U. Kister, C. Tominski, R. Dachselt, and H. Schumann. *Mapping Tasks to Interactions for Graph Exploration and Graph Editing.* Poster at IEEE Conference on Information Visualization (InfoVis). 2015.

[Gla+15c] S. Gladisch, U. Kister, C. Tominski, R. Dachselt, and H. Schumann. *Mapping Tasks to Interactions for Graph Exploration and Graph Editing on Interactive Surfaces.* Tech. rep. University of Rostock, Technische Universität Dresden, 2015.

2.1. Terms and Definitions

The starting point of graph editing and graph visualization is the data. According to [War04], data can be divided into *entities* describing objects of interest (e.g., persons), *relationships* describing the structure of entities and relating them to one another (e.g., friendship between persons) and *attributes* associated with objects and relationships describing their properties (e.g., age of a person). A *graph* is a mathematical concept that is able to represent such entities and their relationships. This concept is important in science and has a wide applicability. Graphs are mainly used for two purposes. Firstly, to represent data in a more understandable way and secondly, to solve problems with the help of algorithms from *graph theory*. In this thesis, the former purpose is of interest.

In the most common sense, a *graph* can be formally defined as an ordered pair $G = (V, E)$ where V denotes a finite set of entities, often called *nodes* or *vertices* and E denotes a set of relations or connections between the objects, often called *edges* or *links*. This definition refers to the definition of a *simple graph* in [Bra94]. The nodes of the graph can represent entities of a dataset, where the edges can be used to represent relationships between entities. Although the term *network* is used differently in literature, it is often associated with the term graph as defined above. Hence, both terms are used synonymously in the following.

Unless otherwise stated, the following definitions are obtained from [Bra94]. If a graph G is undirected, every edge $e \in E$ can be defined as a two-element subset $\{u, v\}$ of V. If G is directed, E is a subset of $V \times V$ [Die06].

An edge e connecting the nodes u and v is considered to be *incident* to u and v, whereas u and v are considered as *adjacent*. The *neighbors* of a node v are all nodes adjacent to v. In an undirected graph, the number of all neighbors of a node v is called *node degree*.

A graph $G_1 = (V_1, E_1)$ is called *subgraph* of $G_2 = (V_2, E_2)$ if $V_1 \subseteq V_2$ and $E1 \subseteq E2$ [Bat+98]. A *clique* is a particular subgraph of an undirected graph, where each pair of nodes is connected by an edge.

An *undirected path* is a sequence $(v_0, v_1, \dots, v_h)$ of nodes of G, such that $\{v_i, v_{i+1}\} \in E$ for $0 \leq i \leq h-1, h \in \mathbb{N}$. If $v_0 = v_h$, the path is called *cycle*.

A graph is called *connected*, if there exists an undirected path between every pair of nodes $u, v \in V$, with $u \neq v$.

A *tree* is a connected graph with no cycles.

A *multivariate graph* is a graph $M = (V, E, A, W)$ with a set of nodes V and a set of edges E as introduced before. Additionally it comprises p node attributes $A = \{A_1, \dots, A_p\}$ and q edge attributes $W = \{W_1, \dots, W_q\}, p, q \in \mathbb{N}$. Every node attribute A_i represents a column in a table $A = (a_{ij})(i \in [1, |V|], j \in [1, p]$ and contains one attribute value per node. $a^i = (a_{i1}, \dots, a_{ip})$ describes all attribute values for node i given that there is no missing data. Edge attributes W are defined analogously. This definition was adapted from [KPW14].

In this work, *qualitative* and *quantitative* node and edge attributes are considered. It should be mentioned that the terms qualitative and quantitative attributes are used differently in literature. A qualitative attribute usually has a value range of finite categories.

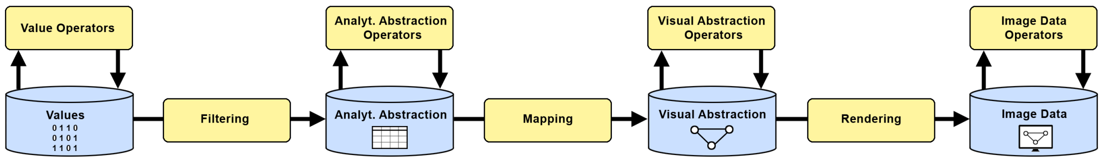


Figure 2.1.: The *data-state-reference model* describing the data transformation process for visualizing data as a pipeline consisting of four data stages with stage operators and three transformation operators [CR98].

Such attributes can be further classified into *nominal* data (no order between values exist) or *ordinal* (an order between values exist) data. An example for a nominal value range is $\{"male", "female"\}$, whereas $\{"good", "neutral", "bad"\}$ is an example for an ordinal value range. A quantitative attribute usually has a finite or infinite numeric value range. In this thesis *discrete* value ranges like $\mathbb{N}$ and *continuous* value ranges like $\mathbb{R}$ are of interest.

In the following the term *graph structure* refers to the set $V \cup E$. The term *graph attributes* refers to the set $A \cup W$. Graphs can also have temporal as well as spatial context which further adds complexity to the data [Had14]. These aspects are beyond the scope of this work.

2.2. Graph Visualization

Graph visualization facilitates the analysis of huge amounts of data by representing them visually. The reason for that is the human cognition: "Visual displays provide the highest bandwidth channel from the computer to the human. Indeed, human acquire more information through vision than through all of the other senses combined" [War04]. Moreover, several pieces of information can be communicated simultaneously, whereas numbers and written language have to be read sequentially [Ber83]. In order to develop a concept for additionally supporting graph editing with these visual representations, they must be investigated first. For that, a basic understanding of the general visualization process is needed.

2.2.1. Data-State-Reference Model

From a conceptual perspective, data visualization is a step-wise process which transforms raw data into a visual representation. For describing this process, different models based on the *visualization pipeline* from Haber and McNabb [HM90] have been developed [JD13; SB04; CMS99]. One comprehensive and established model is the *data-state-reference model* from Chi and Riedl [CR98]. In the *data-state-reference model*, which is illustrated in Figure 2.1, data is processed through four data stages with stage operators and three transformation operators. The data stages are:

Values depicting the raw input data.

Analytical Abstraction i.e., filtered data enriched with meta data.

Visual Abstraction constituting abstract geometrical representations of the data.

Image Data depicting the result of the visualization, i.e. pixels of an image.

Every stage comprises stage operators which are used to transform the data without changing their current structure. To transfer data from one stage to another, the following transformation operators are used:

Filtering operators transform values into analytical abstractions. This includes applying smoothing filters or interpolating missing values or correcting erroneous values for example. Although this step is usually carried out by the computer automatically, it has been realized that sometimes human intervention via interaction is needed [Kan+11]. One example is the correction of erroneous data. "An analyst might assess whether a value is high because of some error (e.g., entered into the database incorrectly) or because it accurately reflects a real world occurrence" [Kan+11].

Mapping operators convert numerical data from the analytical abstractions stage into visual marks (geometric primitives like points, lines, polygons) with a certain appearance (e.g., color, area, texture) in the visual abstractions stage.

Rendering operators transform geometric data from the visual abstractions stage into pixels of the image stage, which can be displayed on a screen.

The visualization of graphs is challenging due to the complex characteristics of the underlying data. "In many cases, given the limits of human perceptual and cognitive abilities, it is impossible to clearly show all the information at once in a useful form" [KPW14]. For that reason, most graph visualization approaches put emphasis on specific data characteristics. Approaches for visualizing the graph's structure or the graph's attributes are two important categories. The following sections provide an overview of existing approaches within these categories.

2.2.2. Visualization of Structure

The aim of visualization approaches with emphasis on graph structure is to generate a visual representation that clearly communicates nodes and edges. According to a State-of-the-Art report for the visual analysis of large graphs [Lan+11], there are four major categories of graph representations: node-link representations, matrix representations, implicit representations and hybrids of these. Each category is described below.

2.2.2.1. Node-Link Representations

The node-link representation is by far the most common visual graph representation and used in many application areas. The reason for that might be that this kind of representation reflects the mental image of a graph in an intuitive way and is, for this reason, easy to understand. In this representation, nodes are mostly depicted as circles or rectangles in 2D (or sometimes as small spheres or cuboids in 3D) and edges between them are often represented as lines.

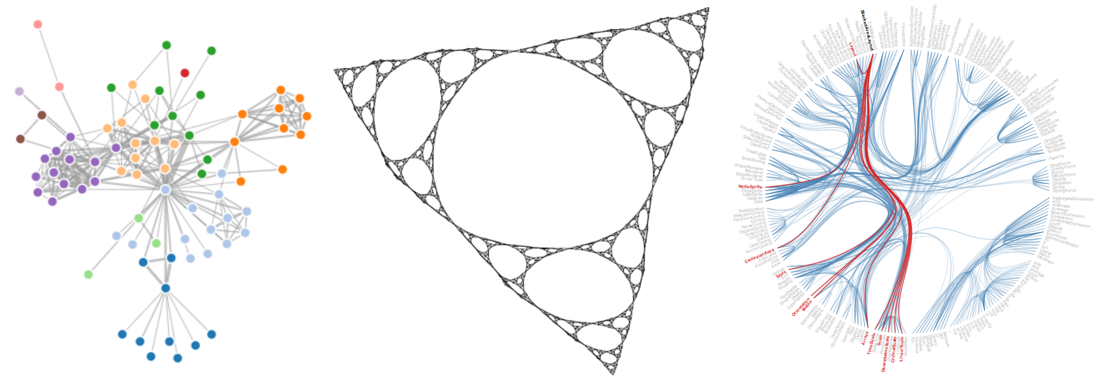
A major challenge is the layout of nodes and edges in these visual representations. A bad layout easily leads to visual clutter with lots of edge crossing and overlaps and thus decreases readability enormously. There are two main approaches to deal with this problem. The first approach is to layout nodes and edges manually. This can be done for small graphs and is supported by most diagramming tools. The second approach – the use of automatic layout algorithms – comes into play for larger graphs. The research field of *graph drawing* provides a vast amount of algorithms for computing node positions and edge routes according to aesthetic criteria (e.g., number of edge crossings, uniformity of edge lengths and symmetric placement of nodes). Well-known examples are force-directed [FR91; Ead84], algebraic [HK04; KCH03; Hal70] and multi-level algorithms [Hac05] for placing nodes. Edge-bundling [Zho+13] as well as poly-line and orthogonal edge routing [MSW14; WMS09; WMS05] are examples for routing edges. Although most of these algorithms prioritize specific aesthetic criteria, the problem of finding an optimal layout often remains NP-complete [DPS02]. For that reason, most of these algorithms provide approximations of the optimal solution only. Figure 2.2 illustrates layout results of selected algorithms. With increasing number of edges, the layout quality usually decreases. This can lead to "hairball" like node-link representations in the worst case.

A special case are node-link representations for graphs with geographic reference where nodes or edges may have predefined positions or routes. An example is a diagram of a train network on a map where nodes depict stations and edges represent train routes.

Assuming that a suitable layout for a given dataset can be computed, node-link representations provide an overview of the overall structure of a sparse graph[1] and support path-related tasks well (e.g., finding a path between two given nodes) [KEC06; GFC04]. The effectiveness of different node-link representations has been evaluated concerning specific structure-based tasks by measuring task performance such as response time and accuracy [Xu+12; Bur+12; HW09; Hal+08; HHE06; GFC04; Pur98], as well as cognitive load [HEH09]. These evaluations revealed that the size of the graph (concerning the number of nodes and edges), the layout and the visual mapping have significant impact on the representations' effectiveness concerning specific tasks.

Node-link representations have been applied in various domains. The visualization of co-authorship networks [Abe+14], social networks [Gou+12] and biological networks [CS04] are examples. More information concerning node-link representations can be found in [GFV13; Bat+98].

[1]A *sparse graph* is a graph in which the number of edges is much less than the possible number of edges.



(a) (b) (c)

Figure 2.2.: Results of selected layout algorithms for node-link representations: (a) force-directed layout of a graph based on character co-occurrences in a novel [Bos15a], (b) the multi-level algorithm presented in [Hac05] applied to a graph based on recursively subdivided triangles, (c) *Hierarchical Edgebundling* [Hol06] applied to a graph based on class dependencies in a software hierarchy.

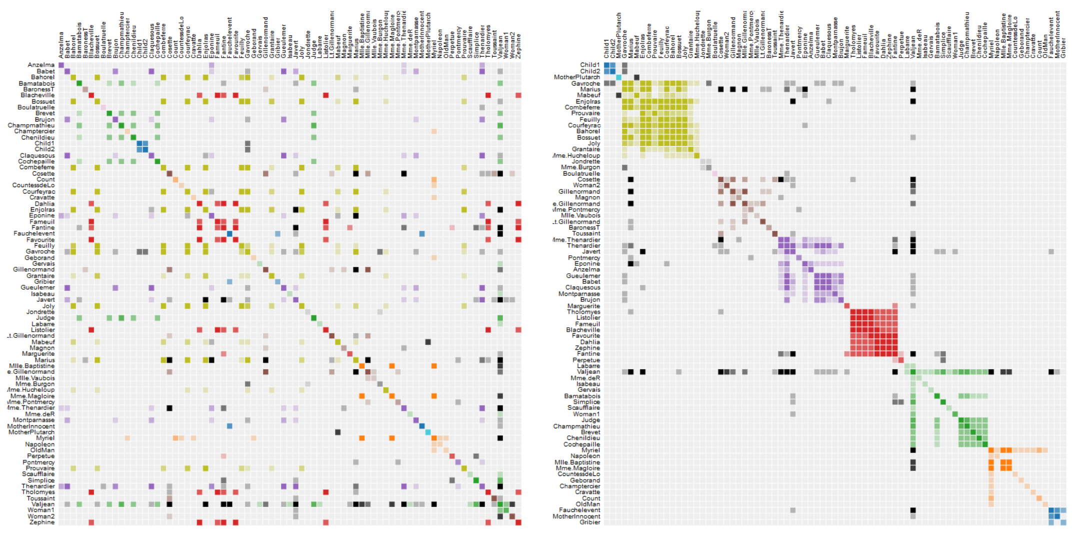
2.2.2.2. Matrix Representations

Matrix representations visualize a graph's adjacency matrix and provide a uniform and structured view with emphasis on the graph's edges. Each node is represented by a column and a row within a matrix. Each cell (i, j) of that matrix depicts whether an edge exists between the node of row i and column j. In this way, the edges are represented without edge crossings or overlaps, which guarantees edge visibility at all times and avoids ambiguities [SM07]. Moreover, existing *and* non existing edges are visualized at the same time. Such matrices are able to represent directed and undirected graphs. In the latter case, the representation becomes symmetric.

The readability and communication of structural features, especially with dense graphs[2], depends on the ordering of the matrix' rows and columns [Ber83]. Similar to node-link layouts, different algorithms for matrix ordering exist. Examples can be found in [Elm+08; MML07; HF06]. Figure 2.3 illustrates two examples of different matrix orderings for the same dataset. With an appropriate ordering, matrix representations are able to reveal cliques, non-existing edges and nodes with many outgoing but no incoming connections. In the latter case, this is visible as a horizontal or vertical line in the matrix and could indicate a spammer in a communication network for example [Had14].

Matrix representations have been compared to node-link representations according to their effectiveness concerning a set of structure-based tasks [GFC04]. With regard to readability, response time, and number of correct answers concerning most of these tasks, the results showed that matrices outperform node-link representations in case of large

[2] A *dense graph* is a graph in which the number of edges is close to the maximal number of edges.



(a) (b)

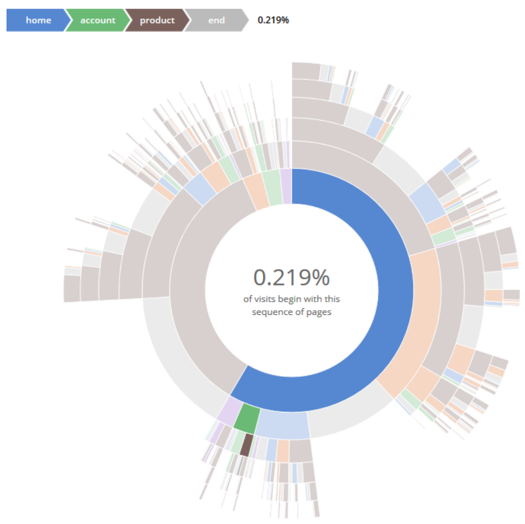
Figure 2.3.: Different orderings applied to a matrix representation of a graph based on character co-occurrences in a novel: (a) alphabetical ordering, (b) ordering according to detected communities [Bos15b].

or dense graphs. Due to their structure, matrix representations are rarely practical for identifying paths along several nodes. To alleviate this issue, visual links for certain paths have been superimposed [SM07; HF07], e.g., for the shortest path between two selected nodes. Furthermore, the space needed for matrix representations rises quadratically with the number of nodes [Lan+11; KEC06]. Hence, when screen space is limited, as for example in mobile applications, matrix representations might be not suitable. To tackle this problem, zoom and aggregation techniques have been proposed [Elm+08; AH04; Ham03]. Moreover, matrix visualizations are rarely useful for sparse graphs because most of the screen space is left blank.

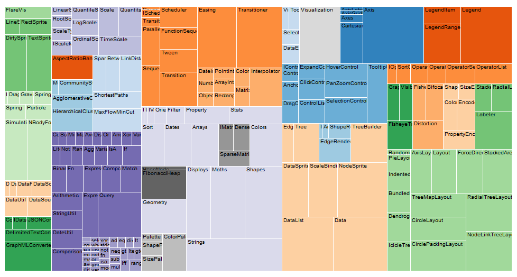
Application examples for matrix representations are the exploration of co-authorship networks [Hen08], social networks [HF06], and protein-interaction networks [Elm+08]. For further information concerning matrix representations, the interested reader is referred to the comprehensive overview provided in [Hen08] or [Sch04].

2.2.2.3. Implicit Representations

Implicit representations are usually limited to trees and often cannot be used for general graphs. For the sake of completeness, they are described too. Implicit representations have in common that the tree's edges are represented implicitly according to the layout of the nodes which are represented explicitly (i.e., with individual visual marks). Common layout strategies for nodes use overlap, adjacency or inclusion to communicate edges.



(a)



(b)

Figure 2.4.: Two examples of implicit representations: (a) a *Sunburst* visualizing visit sequences of a website [Rod15], (b) a *Treemap* visualizing the package hierarchy of a software toolkit [Bos15c].

They are also often referred to as space-filling representations because nodes are arranged tightly in order to use drawing space efficiently. This way, these techniques are capable of presenting a compact overview of even large trees. Similar to other graph representations, finding an appropriate layout is challenging. This includes packing the available drawing space, which is NP-complete by itself, and considering different layout criteria (e.g., an aspect ratio of 1 for the regions of nodes [Wat05]) at the same time.

Various implicit representations have been invented. Common ones include *Treemaps* [Joh93; Shn92; JS91] that use inclusion of nodes to represent their connection, *Sunbursts* [SZ00] or *Icicle Plots* [KL83] where connected nodes are placed adjacent, or *Beamtrees* that utilize node overlaps to communicate edges [HW02]. Figure 2.4 presents two examples. Further examples of implicit representations can be found in [Sch11].

When visualizing directed trees, implicit representations using adjacency to communicate the tree's edges should be avoided because edge direction often cannot be encoded. Some studies exist in literature that evaluate specific implicit representation according to certain tasks. The study presented in [AK07] confirmed that *Treemaps* can support quantification tasks on nodes effectively. However, it has been suggested that so-called *Steptrees* and *Beamtrees* can outperform *Treemaps* concerning further structure-based tasks [BCS04].

A common application example of implicit representations is the visualization of hierarchical file directories [HW02; SZ00; JS91].

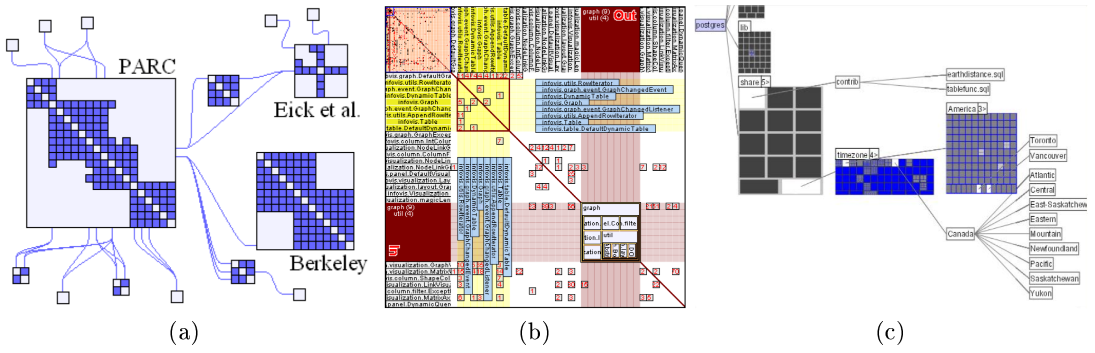


Figure 2.5.: Three different hybrid representations: (a) a *NodeTrix* representation (node-link + matrix representation) visualizing a co-authorship network from information visualization [HFM07], (b) an *Embedded Treemap* representation (matrix + implicit representation) visualizing links between source code modules [RMF09], (c) an *Elastic Hierarchy* representation (node-link + implicit representation) showing the hierarchical directory structure of a database [ZMC05].

2.2.2.4. Hybrid Representations

Hybrid representations are a combination of approaches from the previous sections to better reflect subsets of the graph with specific characteristics. These representations combine the advantages of individual approaches but increase the complexity of the visualization and the associated interaction. As illustrated in Figure 2.5, existing approaches combine matrix and node-link representations (e.g., *NodeTrix* [HFM07]), matrix and implicit representations (*Embedded Treemaps* e.g., [RMF09]), as well as implicit and node-link representations (e.g., *Elastic Hierarchies* [ZMC05]).

NodeTrix for example is well suited for graphs that are locally dense, but globally sparse [HFM07]. Sparse subsets of the data are communicated with node-link representations to better reflect the overall structure of the graph, while matrices are used to better support the analysis of dense subsets.

A challenge with hybrid approaches is to decide which representation to use for which subset of the data. While automatic approaches exist [AMA07], some approaches leave this decision to the user and allow for an interactive switch [HFM07].

Such hybrid representations have been used to visualize social networks [HFM07], software architectures [RMF09] and file directories [ZMC05] for example.

2.2.3. Visualization of Attributes

The major challenge of multivariate graph visualization is to understand the interplay between structural properties of the network and its associated attribute data [KPW14]. For that reason, a lot of visualization approaches emerged that adapt approaches of the previous section to communicate the graph's structure *and* the graph's attributes.

These approaches can be divided into approaches using representations with attribute-dependent and attribute-independent layouts.

2.2.3.1. Representations with Attribute-Dependent Layouts

These kinds of representations have in common that their spatial layout encodes graph attributes. Hence, graph visualization approaches generating such representations use a mapping from attributes to the visual variable position.

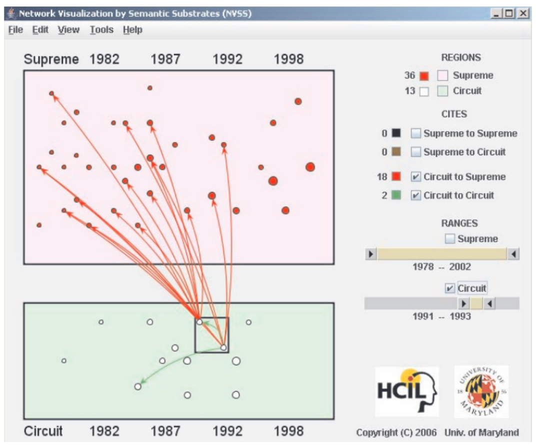
Node-Link Representations: There are two major approaches using attribute-dependent layouts in node-link representations. Approaches based on *Semantic Substrates* [Rod+11; SA06] place nodes in non-overlapping regions according to one qualitative node attribute. When proportionally sized regions are used, users get an idea of the relative cardinality of nodes in each region. Another advantage is that users can distinguish edges that cross from one region to another quickly [SA06]. Figure 2.6(a) depicts an exemplary semantic substrates layout of a graph based on court proceeding data, where nodes represent court cases, edges represent citations between court cases and regions depict specific courts. A closely related approach for visualizing node clusters concerning node attribute values is presented in [Jus+14]. In this approach, a circular layout of clusters and nodes within clusters is used. This idea can be transferred to semantic substrates representations by using a circular layout for regions and nodes within regions.

The second approach is based on scatterplot layouts that place nodes according to two previously selected node attributes in 2D-Space. This often leads to scatterplot like representations as exemplified in Figure 2.6(b). Examples for this approach are described in [EW14; Bez+10; Wat06; WT06]. While such representations may help to generate hypotheses about how connections relate to multiple attributes [Wat06], they may cause a reduced readability of the graph's structure due to overlapping nodes and edge clutter. To address this issue, overlapping nodes can be placed around a circle centered on their original position [Bez+10]. Another strategy is to present node or edge aggregations as demonstrated in [EW14].

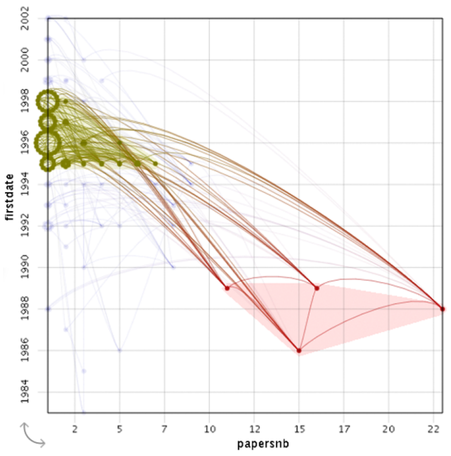
Scatterplot layouts cannot be used for graphs where node positions are predefined, e.g., by geographic reference.

A first approach for an attribute-dependent node-link layout concerning a single edge attribute is introduced in [JKZ13]. In this layout, nodes are positioned in a way so that edge lengths encode a selected edge attribute.

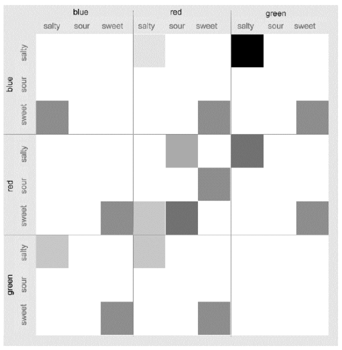
Matrix Representations and Implicit Representations: Although matrix and implicit representations provide options to map node attributes to node positions (e.g., the position of a row or column of a matrix representation or the horizontal or vertical position of a node in a *Treemap*), only few approaches utilize this strategy. In [Wat06], a matrix representation is presented that uses a layout according to two node attributes. Although the author states that this could be helpful for experienced users, examples are given where node-link representations with applied scatterplot layouts clearly outperform this


Network Visualization by Semantic Substrates (NVSS)
File Edit View Tools Help
Supreme 1982 1987 1992 1998
Circuit 1982 1987 1992 1998
REGIONS
36 Supreme
13 Circuit
CITES
0 Supreme to Supreme
0 Supreme to Circuit
18 Circuit to Supreme
2 Circuit to Circuit
RANGES
Supreme
1978 – 2002
Circuit
1991 – 1993
HCIL
Copyright (C) 2006 Univ. of Maryland

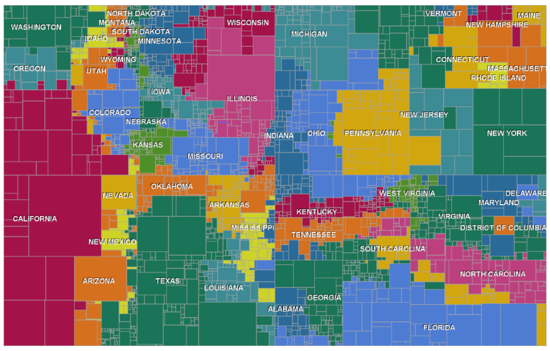
(a)



(b)



(c)



(d)

Figure 2.6.: Four different representations for visualizing graph attributes with attribute-dependent layouts: (a) *Semantic Substrates* layout of a node-link representation placing nodes in different regions according to a qualitative node attribute [SA06], (b) *scatterplot Layout* of a node-link representation placing nodes according to two node attributes in 2D-Space [Bez+10], (c) layout of a matrix representation according to two node attributes [Wat06], (d) layout of an implicit representation (*Treemap*) considering geospatial constraints [Gho+15].

approach. Whether a mapping to positions in matrix representations can be reasonable or not remains an open research question.

Concerning implicit representations, there are currently no approaches using a direct mapping from node attributes to the visual variable position. This is due to the fact that a direct mapping to position would often conflict with the layout strategies used to communicate the graph's edges (overlap, adjacency, or inclusion). However, there are approaches for *Treemaps* that use node attributes as a specific criterion for their positioning. *Spatially dependent Treemaps* [Gho+15; WD08; Man+07; Kei+05; Hei+04] are one example. These approaches incorporate geospatial constraints of the nodes into the layout process (e.g., the geo-location of nodes). Application examples for these approaches are the visualization of hierarchically subdivided territories as illustrated in Figure 2.6(d) [Gho+15; WD08; Man+07], autonomous systems [Man+07] and internet traffic [Kei+05; Man+07].

Another recent approach taking node-attributes into account for a *Treemap* layout is presented in [Dua+14]. In this solution, the nodes are placed close to each other if they share similar attribute values.

2.2.3.2. Representations with Attribute-Independent Layouts

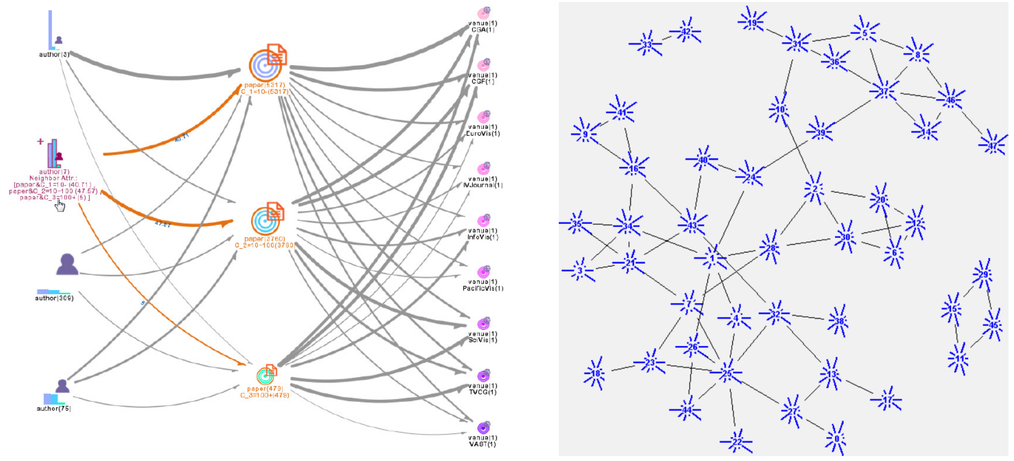
Representations with attribute-independent layouts apply a common structure visualization and either encode attributes with additional visual variables like color, shape or brightness, or integrate glyphs in node or edge representations.

While there are lots of different options for visual variables [Mac86], their applicability strongly depends on the base representation and the data. Furthermore, visual variables differ in their applicability for attribute types as well as their effectiveness can interfere pairwise. An example for visual variables introducing some interference is area and color, as color is harder to perceive for small objects [Mun14]. For that reason, visual variables have to be selected carefully. In the same way, the suitability of integrating glyphs strongly depends on the size of the node and edge shapes as they have a strong impact on the glyphs' readability.

Node-Link Representations: As node-link representations visualize nodes and edges with explicit primitives, their respective representation can be varied according to associated attributes. For the encoding of qualitative node attributes (e.g., the node type of an author-paper-venue heterogeneous network [Shi+14]), different node shapes are often used [DS13; Sta+07; Mad+05]. Figure 2.7(a) provides an example. Quantitative node attributes (e.g., the post count of a user in a forum [HP11]) are mostly communicated with varying color, brightness or size [HP11; Wat06]. For representing edge attributes similar approaches have been developed. The edge shape can be used to encode a qualitative attribute, for instance the type of relation. The UML standard for class diagrams follows this strategy. Moreover, color, width or brightness can be used to communicate quantitative edge attributes [Wat06; Roh+12].

Since node-link representations are commonly two-dimensional, it is also possible to encode a node or edge attribute by height resulting in a 2.5D representation. In this context, height fields have been utilized to represent node attribute values as elevations above nodes [Xu+07].

The integration of glyphs has been used in node-link representation in various ways. Examples are the integration of bar charts [ME09], glyphs similar to parallel coordinates [JDK10] or the substitution of nodes with stick figures as illustrated in Figure 2.7(b) [WT08]. A problem that may arise with this approach is reduced readability, caused by small node-sizes, node overlaps or edge crossings. To tackle this problem, an interactive



(a) (b)

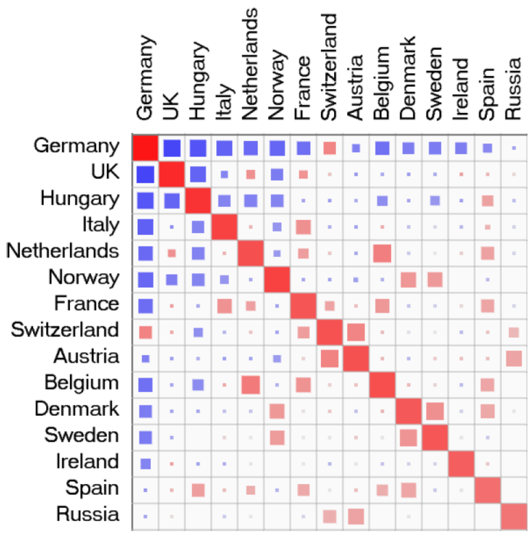
Figure 2.7.: Two different approaches for encoding node attributes in node-link representations: (a) varying values of a node attribute encoded by varying node shapes [Shi+14], (b) multiple node attribute values encoded with integrated glyphs [WT08].

lens that restricts the integration of glyphs to a local region of interest can be utilized [JDK10]. As the integration of glyphs subdivides a given shape into smaller shapes that comprise individual visual variables on their own, this concept cannot be used for edges that are presented with lines.

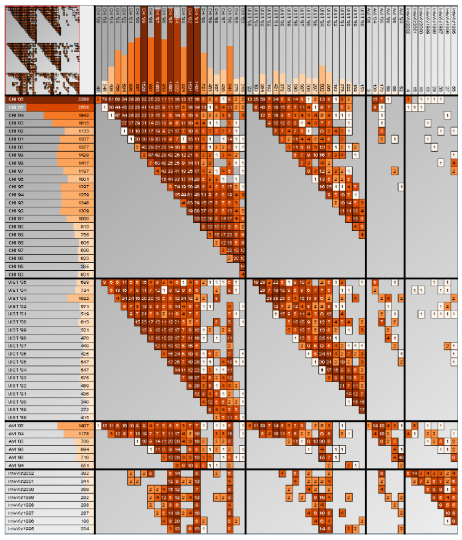
Matrix Representations: Due to the explicit visualization of edges in matrix representations, their appearance can be varied according to associated attributes. At the same time, the regular grid structure of matrix representations limits the options to encode edge attributes by varying shape, size and orientation. Considering this, in [HSD09] the size of a square shape drawn inside each cell is modified according to an edge attribute. An example of such a matrix representation is shown in Figure 2.8(a).

A common approach is to use color or brightness to encode edge attributes, as exemplified in Figure 2.8(b). This example visualizes a conference citation network and encodes the frequency of citations with varying color and brightness [Hen08]. Another solution is to integrate glyphs in matrix cells, for example small bar charts as presented in [Elm+08].

For the encoding of node attributes there are only a few options available. Because row and column sizes, orientations and shapes are specified by the regularity of the grid, these visual variables cannot be used. When nodes are additionally represented with labels attached to rows and columns, further options become available. As illustrated in Figure 2.8(b), the background-color of the labels can be used to encode a selected node attribute for example.



(a)



(b)

Figure 2.8.: Two different approaches for encoding attributes in matrix representations: (a) two edge attributes encoded by varying color and sizes of squares drawn within matrix cells [HSD09], (b) one edge and one node attribute encoded with varying color and brightness of matrix cells and node labels [Hen08].

Implicit Representations: Since implicit representations do not visualize edges explicitly, edge attributes cannot be visualized by varying edge appearance. Nodes in turn are represented explicitly and facilitate the encoding of node attributes.

Although the layout of implicit representations is usually optimized to communicate structural aspects of the tree, there are approaches encoding a node attribute by size, which also influences node positions. One prominent example can be found in the paper introducing *Treemaps* [JS91]: Johnson and Shneiderman visualize the space consumption of files in a file directory with a *Treemap*, where the nodes' sizes represent file sizes [JS91]. In another example [WL07], classes of a software system are visualized with nodes of a *Treemap*. In this example, node width and node height communicate the number of class methods and class attributes.

The visual variables color, texture and brightness can generally be used to encode node attributes [WD08; HVW05]. The example illustrated in Figure 2.9(a) visualizes a file hierarchy and encodes varying file age by varying colors. Another way to communicate a node attribute is to use a 2.5D implicit representation encoding a node attribute by height [SL10; WL07; BCS04]. As illustrated in Figure 2.9(b), this can lead to occlusion and demands for additional interaction techniques like rotation or the use of transparency.

The integration of glyphs to visualize node attributes was also considered for implicit representations. Approaches integrating bar charts can be found in [KMT12] and [ME09]. A problem with the integration of glyphs in implicit representations is the different aspect

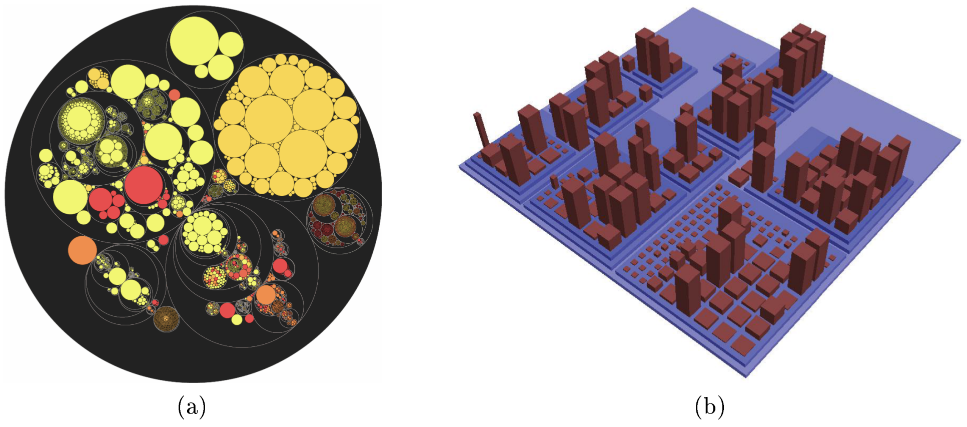


Figure 2.9.: Two different approaches for encoding node attributes in implicit representations: (a) a circular *Treemap* encoding a node attribute by color [Wet03], (b) a *Treemap* encoding a node attribute by height [WL07].

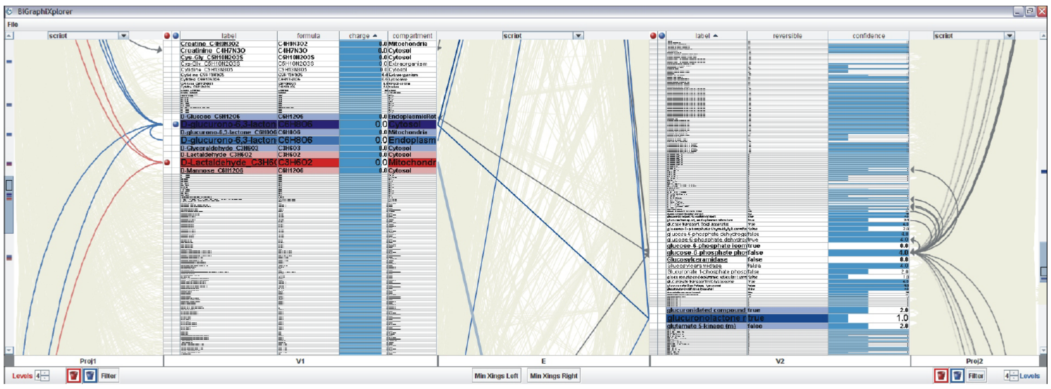
ratio of the node shapes. This results in glyphs of varying width and height which are hard to read, hinder comparison and may cause wrong conclusions.

Some studies exist in literature evaluating specific implicit representation according to their effectiveness for supporting certain tasks. For example, the study presented in [WTM06] confirmed, that *Treemaps* are well-suited for supporting node-attribute-based tasks involving judgment of similarity and homogeneity. The effectiveness of particular implicit representations using adjacency has been indicated concerning node-attribute-based and structure-based tasks [BN01; Sta+00].

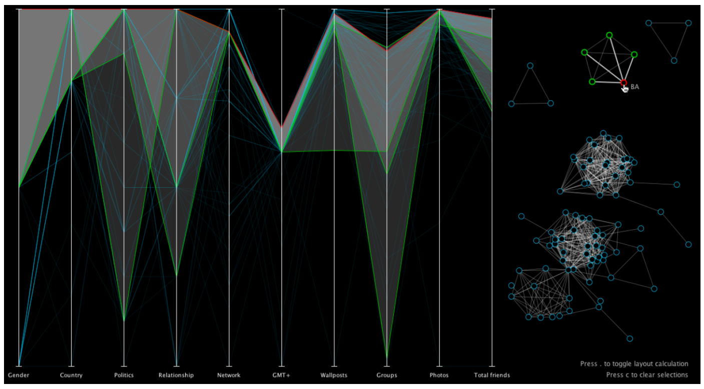
2.2.3.3. Further Approaches

There are two further kinds of approaches that do not match the above categories: Tabular representations and approaches using representations for multivariate data in an interlinked view setting.

Tabular Representations with Visual Links: The idea of tabular graph representations with visual links is to represent specific data characteristics as tables and to link rows of individual tables visually to communicate a graph's structure. Often, a partitioning of the set of nodes is represented with individual tables, whereas each row of each table visualizes a specific node [PW08; Sch+08; Sta+07]. Edges are visualized as lines connecting nodes similar to node-link representations. Graph attributes can be represented in various ways. There are approaches using different fillings of table cells, or labels in cells of separate columns [PW08; Sch+08]. Edge attribute values have been represented for instance with varying color of the connected rows [Sta+07].



(a)



(b)

Figure 2.10.: Two further examples for visualizing graph attributes: (a) a tabular representation of a biological network showing node attributes as columns of a table. Individual attribute values are represented with labels and fillings [Sch+08]. This figure was taken from [Had14]. (b) A coordinated view setting consisting of a *Parallel Coordinates Plot* and a node-link representation showing a social network [SHQ08].

An advantage of tabular representations is that they are mostly equipped with filtering and sorting mechanisms, which may help in finding attribute-correlations. To reduce edge crossing, appropriate ordering mechanisms can be applied. An application example of tabular graph representations is the visualization of bipartite biological networks as presented in [Sch+08] (see Figure 2.10(a)).

Representations for Multivariate Data: There exist several general approaches for visualizing multivariate data [Spe07] that can be used to visualize a graph's attribute values. According to [KPW14], they can be roughly categorized in point-based approaches (e.g., *Scatterplot Matrices* [CM88]), axis-based approaches (e.g., *Parallel Coordinate Plots* [ID90]), glyph-based approaches (e.g., *Chernoff Faces* [Che73]) and pixel based-approaches (e.g., *Recursive Patterns* [Kei02]). A common approach is to use these representations side-by-side with a representation communicating the graph's structure. An example is presented in Figure 2.10(b).

2.3. Graph Editing

Visualizing graphs in order to communicate their major characteristics is only one aspect when dealing with graphs. Often, there is also the need to change data of graphs. For that reason, graph editing is investigated in the following. The term graph editing is used differently in literature, requiring a short definition to refer to in this thesis:

▷ *Graph editing* is the process of altering at least one of the graph's set of nodes, set of edges or set of attribute values associated with nodes or edges.

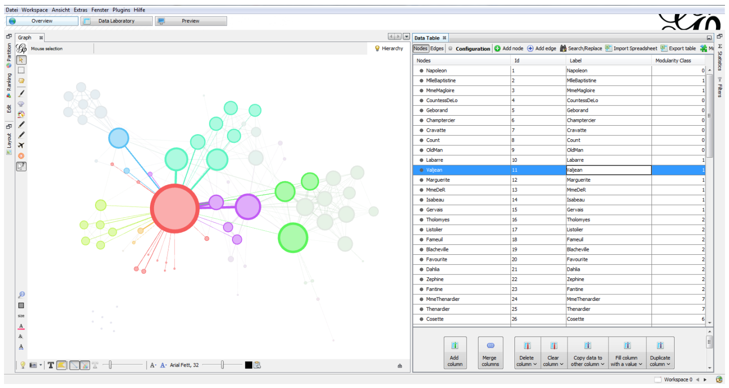
Hence, a prerequisite for graph editing is a well-defined graph. Then, graph editing can be done either automatically (if the changes to be made are known), interactively or as a combination thereof. As this thesis focuses on interactive editing the term *graph editing* refers to this approach in the following.

The reasons for graph editing can be manifold. They incorporate the maintenance of graphs to ensure the correct status of the data (e.g., updating a co-authorship graph according to newly published papers), graph corrections or cleansing to provide a basis for further goals (e.g., correcting erroneous attribute values may be necessary prior to analysis [Kan+11]), graph creation (e.g., a newly created UML-diagram in software engineering), or the alteration of the data to create and analyze what-if-scenarios (e.g., inserting an edge between two certain nodes for analyzing the change of the graph's overall connectedness).

Existing approaches for graph editing can be roughly categorized in approaches using syntactical tabular representations of the graph and approaches using visual node-link representations. These approaches are reviewed next.

2.3.1. Graph Editing Using Tabular Representations

This category of approaches can be further divided into approaches using visual representations and tabular representations in an interlinked view setting, and approaches that rely on tabular representations exclusively. The former approach usually allows users to select nodes or edges in the visual representation which are then highlighted in the tabular representation (or the other way around). The editing can be done in the tabular representation through text input and is often limited to changing attribute values of nodes and edges. This editing strategy is typically employed in graph visualization systems. Examples are *Cytoscape* [Sha+03], *Tulip* [Aub04] and *Gephi* [BHJ09] as illustrated in Figure 2.11. Although such interlinked view settings support the user in locating selected nodes or edges across views, they separate graph analysis and editing.



Nodes	Id	Label	Modularity Class
Napoleon	1	Napoleon	0
MlleBaptistine	2	MlleBaptistine	1
MmeMagloire	3	MmeMagloire	1
CountessDeLo	4	CountessDeLo	0
Geborand	5	Geborand	0
Champtercier	6	Champtercier	0
Cravatte	7	Cravatte	0
Count	8	Count	0
OldMan	9	OldMan	0
Labarre	10	Labarre	1
Valjean	11	Valjean	1
Marguerite	12	Marguerite	1
MmeDeR	13	MmeDeR	1
Isabeau	14	Isabeau	1
Gervais	15	Gervais	1
Tholomyes	16	Tholomyes	2
Listolier	17	Listolier	2
Fameuil	18	Fameuil	2
Blacheville	19	Blacheville	2
Favourite	20	Favourite	2
Dahlia	21	Dahlia	2
Zephine	22	Zephine	2
Fantine	23	Fantine	2
MmeThenardier	24	MmeThenardier	7
Thenardier	25	Thenardier	7
Cosette	26	Cosette	6

Figure 2.11.: Graph editing using an interlinked table in *Gephi* [BHJ09].

This way the user has to switch between different representations continuously, which may be interruptive. Moreover, due to different data representations, users have to retain a considerable amount of information in short term memory.

Spreadsheet programs like *Microsoft Excel* and data base systems like *Neo4j* with its web interface [Ter15] are examples for approaches that rely on tabular representations exclusively. The editing of the graph's structure or attribute data can either be done through direct text input or query commands using mouse and keyboard interaction. Spreadsheet programs have the advantage that they offer powerful tools to derive values based on existing values (e.g., calculate average). This can be helpful in certain editing scenarios. However, these approaches often do not provide visual representations for communicating the data. For that reason, they should be applied in cases where intended changes are well-known and where further analysis is not required. Replacing undefined values with default values is one example.

2.3.2. Graph Editing Using Node-Link Representations

Whereas interaction techniques for graph visualization are usually applied to modify the visual representation of the data (e.g., graph lenses [Tom+06], tools for layouting nodes [MJ09] or direct manipulation techniques for routing edges [Hen+12]), there also exist techniques for changing the graph's data themselves.

Editing in node-link representations is supported by several systems for desktop environments. Examples are *yEd Graph Editor*, *GoVisual Diagram Editor*, *Visual Paradigm* and *Enterprise Architect*. The oldest ones date back to 1969 [Rob87; TN87; EHS69]. Most of these systems are not limited to graphs and node-link representations, but support diagramming in general. Anyhow, many different types of diagrams are based on graphical objects that are connected with lines, for instance UML-diagrams, business process diagrams, or entity-relationship diagrams, and can thus be interpreted as node-link representations of a graph.

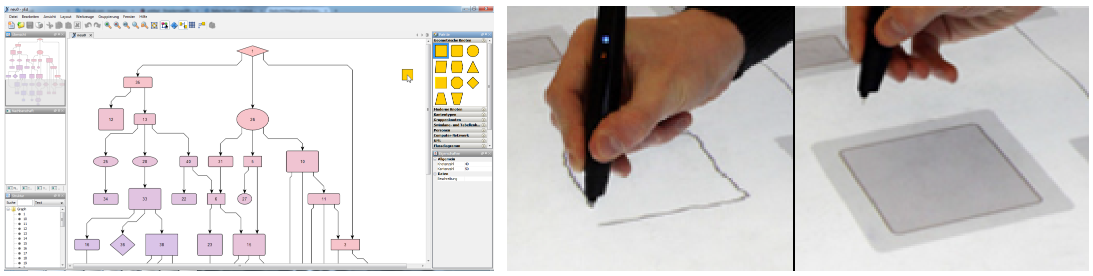

(a) (b)

Figure 2.12.: Exemplary techniques for creating nodes on desktop systems and systems for interactive surfaces: (a) Desktop system: creating a new node by dragging a template from a template palette in *yEd Graph Editor*. (b) System for interactive surfaces: creating a new node by sketching it [FHD10].

The applied editing techniques are quite similar across these systems. As exemplified in Figure 2.12(a), the creation of a node is usually accomplished by dragging a node template from a palette into the workspace or by selecting a node template and clicking on the target position within the workspace. The result is a newly created node with default properties concerning its graphical appearance and default values concerning its attributes. For the insertion of an edge, generally two different techniques are used. In most of these systems, clicking on an edge template in a toolbar invokes a global mode which allows users to insert a new edge by clicking on its source node and dragging to its target node. Sometimes, a second click on the target node is used instead of a mouse-drag. The creation of multiple nodes and edges using these techniques result in excessive cursor movement between template palettes and the workspace. To reduce the user effort, some systems like *Enterprise Architect* or *Visual Paradigm* provide context menus for nodes which can be used to insert neighboring nodes and associated edges automatically.

Moreover, most of these systems support copy and paste operations on selected nodes or subgraphs. This is also mostly done with the help of context menus. The selection can be done with common techniques like selection frames. As inserting and deleting nodes or edges influences the layout of the node-link representation, techniques for adjusting the layout on demand are often provided. This includes techniques for manual repositioning, resizing and aligning nodes as well as edge routes by hand.

The editing techniques described above address desktop environments with classic mouse and keyboard interaction. However, there also exist systems utilizing touch and pen interaction for editing in node-link representations. The systems presented in [GH07; HD06; CGH03; DHT00] enable users to sketch UML-diagrams with pen interaction on e-whiteboards or tablets the same way they would do on paper. In [AN06; AN05], similar approaches for sketching and interpreting hand drawn node-link representations of directed graphs are introduced. A general set of touch and pen gestures for editing data in node-link representations on interactive surfaces is presented in [FHD10]. This gesture set is based on a collection of user-defined gestures from a user study [FHD09]. Figure 2.12(b) depicts an exemplary sketch gesture for creating a new node.

All these approaches have in common that they allow users to edit graphs visually. The used visual representations support the users in understanding the data they are working with. Provided visual feedback upon editing enable the users to perceive that the data has been changed and how. Visual feedback further supports the users in relating data changes to the rest of the data. This is an advantage over editing in syntactical representations as changes to numbers and written language are mostly harder to perceive [Ber83]. Moreover, the visual representation supports graph analysis and editing simultaneously so that cumbersome switches between different representations to accomplish the particular goals are not necessary anymore. A similarity of existing approaches for editing graphs visually is that they concentrate on structural editing in node-link representations. While the usefulness of such manual editing approaches has been demonstrated for moderately-sized graphs being constructed from scratch, they fall short of addressing larger graphs with hundreds of nodes and edges. In such scenarios, computer-based assistance is required.

A general approach addressing data editing in visual representations is presented by Baudel in [Bau06]. This seminal work is the basis of the concept introduced in Chapter 3 and is described next.

2.3.3. General Editing Model

In the work presented in [Bau06], it is argued that the benefits of direct manipulation for data editing are not considered sufficiently in today's graphical user interfaces. For that reason, an approach is presented that extends visual representations for analyzing data by editing facilities that rely on direct manipulation. These editing facilities make use of conventional graphic manipulation techniques found in drawing tools (e.g., tools for moving and resizing objects) for editing the data.

As illustrated in Figure 2.13, the principle is to apply graphical changes to the visual representation of the data which are then back propagated through the stages of the visualization process (see Section 2.2.1) to the raw data and subsequently cause a data change. A precondition is that the visualization process can be inverted. Because data gets transformed and reduced in the visualization process and the transformations are not necessarily bijective, such an inversion can be demanding as described in [FS92].

The set of available graphic manipulation techniques for editing is "deduced" from the data encoding used in the visual representation. Although it is not described explicitly by Baudel, the graphical effect of such techniques must concern one or multiple visual variables used to encode data characteristics. For example, when varying attribute values of data objects are encoded by varying size of graphical primitives, a technique for changing the area of these primitives, e.g., a resizing-technique, can be used. Examples given in [Bau06] include drag and drop techniques for scatterplots and 1D-resizeing techniques for bar charts.

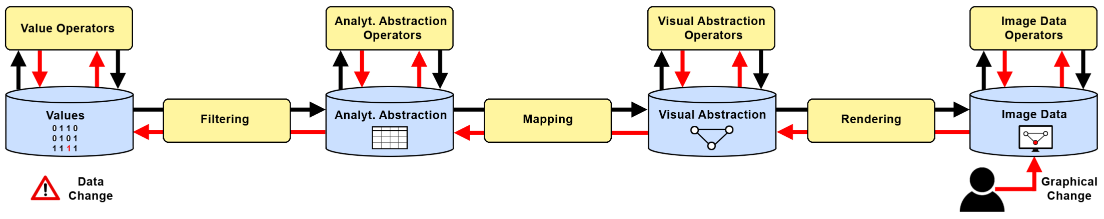


Figure 2.13.: Principle for graph editing described in [Bau06]: A user applies graphical changes interactively which are then back-propagated through the stages of the visualization process and eventually cause a data change. The visualization process is modeled with the *data-state-reference model* [CR98] in this figure.

To describe interrelation of editing tasks, Baudel introduces the *general editing model*. It consists of the following three steps:

- **Adjust** the **representation** in order to match the analysis and editing tasks to be performed. This includes navigational tasks like zoom and pan as well as changing the encoding of the representation.
- **Select data** of interest that is intended to be affected by editing operations.
- **Perform editing** tasks with at least one of five operations: change an attribute value of the selected object(s), clone the selected object(s), delete the selected object(s), add or remove attributes from the selected object(s).

Although it is described that these steps can be performed in an arbitrary order, a selection of objects of interest is always needed before one of the mentioned edit operations can be applied. This is modeled with an activity diagram in Figure 2.14. Editing operations are applied with graphic manipulation techniques. Contrary to the definition of graph editing from the beginning of this chapter, changing, adding and removing attributes of objects (i.e., changing the data definition) is also considered as editing in [Bau06].

With this model, data editing in visual representations becomes possible. However, it must be extended to address the editing of graphs. Currently, this model does not allow inserting edges between existing nodes, nor does it allow creating new graphs from scratch. The reason for this is the set of editing operations used in this model. Moreover, some important tasks like identifying objects of interest are not modeled. How this model can be extended to encompass graph editing is presented in the following chapter.

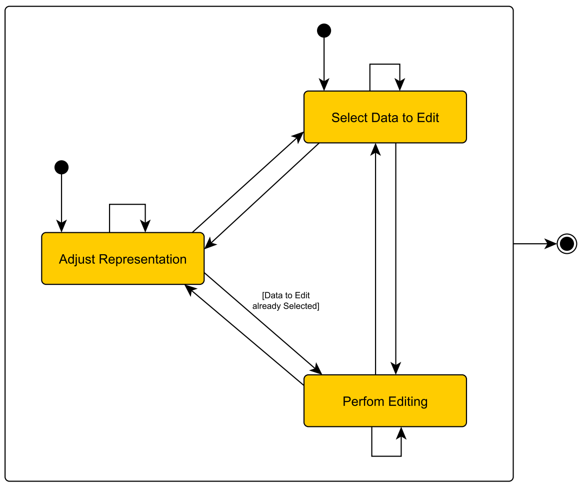


Figure 2.14.: Activity diagram of the *general editing model* presented in [Bau06]. Data editing in visual representations is modeled with three tasks: adjust view settings, select object(s) to be edited, and perform editing operations. It is assumed that a visual representation is given at the beginning of the editing process.

2.4. Summary, Unsolved Problems and Goals

In this chapter, essential terms and definitions for graphs were provided, graph visualization and graph editing were described, and the state of the art concerning both was reviewed. As described in Section 2.2.2, three classes of approaches for visualizing the structure of graphs can be distinguished: approaches using node-link representations, approaches using matrix representations and hybrids thereof. As further described in Section 2.2.3, the majority of approaches visualizing attribute data associated with graphs can be assigned to approaches using representations with attribute-dependent and attribute-independent layouts. While various of these visual graph representations have been used to support analysis, they have been rarely considered for graph editing. For investigating how to additionally use these representations to support visual graph editing, involved tasks must be described first. Thus, one goal of this thesis is:

- Goal 1: The development of a conceptual view on tasks for visual graph editing. This includes describing fundamental tasks involved in visual graph editing as well as modeling their interplay. Moreover existing techniques must be examined to identify tasks that are fully, partially or rather not supported. Chapter 3 deals with this goal.

Existing approaches for graph editing in visual representations are based on node-link representations (see Section 2.3.2). However, their scope of application is limited to structural editing of graphs with moderate size. Editing the structure of larger graphs with established and constraint-based node-link layouts has yet to be addressed. Thus, another goal of this thesis is:

- Goal 2: The development of a novel approach for the structural editing of larger graphs in established and constraint-based node-link layouts. A major issue to be considered is to maintain existing layout constraints and quality criteria and to preserve the user's mental map during editing (Chapter 4).

Although it has been indicated that different representations have different advantages and disadvantages for graph editing [Bau06], matrix representations have not been considered for structural graph editing so far. This leads to the following goal:

- Goal 3: The development of a novel approach for editing the graph's structure in matrix representations (Chapter 4).

Approaches for the editing of the graph's attribute values in visual representations do not exist. For this reason, such editing is still carried out using syntactical, tabular representations. As reviewed in Section 2.3.1, these syntactical representations are not designed for analyzing data and thus require switches to visual representations for accomplishing editing and analysis. Hence, a fourth goal of this thesis is:

- Goal 4: The development of a novel approach for the visual editing of the graph's attribute values (Chapter 5).

A major issue when working with visual representations of larger graphs is that they mostly cannot encode all the data at once. For that reason, navigation techniques like pan and zoom must be used frequently to analyze or edit certain subsets of interest. However, the user is often unaware of where potentially noteworthy data is located, which often leads to a trial-and-error navigation procedure. Thus, goal 5 is to alleviate this issue:

- Goal 5: The development of a novel approach supporting the navigation to interesting data (Chapter 6).

Besides navigation techniques, magic lenses represent another option to get an alternative view on the data in a local region of interest. A lot of individual lens techniques have been proposed to facilitate the visual analysis of data. However, they have not been considered in context of graph editing and a comprehensive overview of such techniques does not exist in literature. This leads to goal 6:

- Goal 6: Provide a comprehensive overview of lens techniques in visualization and investigate lenses for graph editing (Chapter 4 and Chapter 6).

3. A Conceptual View on Tasks of Visual Graph Editing

The previous chapter has shown, that editing in visual representations can be advantageous compared to editing in syntactical representations. However, the full potential of visual editing especially for graphs is currently not exploited. To be able to develop techniques for visual graph editing with regard to both, structure and associated attribute values, it is necessary to investigate the variety of involved tasks. Therefore, this chapter is dedicated to the development of a conceptual view on tasks of visual graph editing. The first Section 3.1 describes categories of tasks involved in visual graph editing and examines their current support. Based on the *general editing model* from [Bau06], Section 3.2 introduces an extended model which describes the interplay of the tasks involved in visual graph editing. The last Section 3.3 summarizes this chapter.

Contents

3.1. Tasks of Visual Graph Editing 31

3.1.1. *Select or Specify Data to Edit* 32

3.1.2. *Apply Edit Operation* 33

3.1.3. *Analyze Data* 34

3.1.4. *Adjust Representation* 36

3.2. A Task-Transition Model for Visual Graph Editing 38

3.3. Summary 43

3.1. Tasks of Visual Graph Editing

In this work, tasks involved to visually edit graphs are considered as activities with specific goals that a user wishes to accomplish by interacting with a visual graph representation of a visualization system. Investigating such tasks is important for two reasons. First, to aid designers in developing novel editing techniques that are tailored to the goals of these tasks. Second, to be able to evaluate systems supporting visual graph editing. The research of tasks involved in visual analysis has a similar motivation [AES05]. In literature, neither tasks of visual data editing in general, nor specific tasks of visual graph editing have been comprehensively described. Thus, the aim of this section is to provide such an overview.

Tasks of visual graph editing can be divided into categories with specific goals. Two fundamental categories already provided by the *general editing model* from [Bau06] are *Select Data to Edit* and *Perform Editing*. These categories comprise tasks for achieving a data change. To achieve this goal, two kinds of information must be provided interactively. *What* data must be edited and *how* they must be edited. Hence, the data to be edited has to be selected (if existent) or specified (if not existent) and edit operations with the desired effect must be applied to them. Because the *Select Data to Edit* category does not encompass the specification of data to be inserted, it is extended to *Select or Specify Data to Edit* in this thesis. The *Perform Editing* category is named misleadingly, since this is the general goal of all tasks involved in visual graph editing. Thus, this category is renamed to the more precise *Apply Edit Operation* in this thesis.

Data analysis with regard to graph editing is another fundamental goal of the user. It is necessary for discovering erroneous or missing data subsets that need to be edited [Kan+11] and for obtaining information about the editing effect. For these reasons, it is clearly a part of visual graph editing and deserves an own task category: *Analyze Data*. In summary, there are three fundamental categories of tasks of visual graph editing:

- *Select or Specify Data to Edit*
- *Apply Edit Operation*
- *Analyze Data*

As the accomplishment of tasks from these categories depends on the visual representation provided to the user, the *general editing model* provides another task category for visual graph editing, which is:

- *Adjust Representation*

This category contains tasks associated with adjusting the representation to enable or ease the accomplishment of *Select or Specify Data to Edit*, *Apply Edit Operation* and *Analyze Data* tasks. Hence, *Adjust Representation* tasks are not considered to be fundamental for visual graph editing but supportive.

In the following, these four task categories along with their containing tasks are described in detail. Moreover, it is examined how these tasks are currently supported with existing techniques.

3.1.1. *Select or Specify Data to Edit*

Select or Specify Data to Edit tasks are important to define the scope of graph editing, i.e. *what* data needs to be changed. Hence, they must be accomplished before an edit operation can be applied. Depending on the intended edit operation to be performed, one can distinguish between two options: i) the intended edit operation affects a subset of a given graph (i.e., updating attribute values or deleting nodes or edges), or ii) the edit operation affects data which is not a subset of a given graph (i.e., inserting nodes or edges). In the first case, an existing subset must be selected by the user. This subset can include nodes and edges referring to the graph's structure, or it can include attribute values of nodes and edges referring to the graph's attributes. This leads to one of the following tasks:

- Select a set of nodes and/or edges to define the scope of structure edit operations.
- Select a set of attribute values from a set of nodes and/or edges to define the scope of attribute value edit operations.

For accomplishing these tasks, the user has to define the subset of the data that needs to be selected. Thus, interaction on the data level is necessary [FS92]. Several techniques have been developed supporting the user in this respect. Most of them provide facilities to mark the graphical primitives corresponding to a data subset of interest (i.e., a set of pixels), which automatically results in a selection of this data subset. A precondition for this principle to work is that the user's input can be traced back one-to-one to a data subset. For that, the visualization process must be traversed backwards, which includes inverting their transformation steps [Bau06; FS92]. Established examples for selection techniques are selections via mouse click, rectangular selection frame or lasso. These techniques are standard for selecting nodes, edges or subgraphs among existing graph visualization systems as well as graph editors. Gestural interaction techniques combining selection and deletion of nodes and edges seamlessly are presented in [FHD10].

When new nodes and/or edges are intended to be inserted into a graph, they must be specified first. This also includes the specification of node and edge attribute values in case of a multivariate graph. Thus, the following task must be performed by the user.

- Specify a set of nodes and/or edges to define the scope of structure insert operations.

The accomplishment of such tasks is usually supported with GUI-dialogues where users can input the characteristics of the data to be inserted. As described in Section 2.3.2, another existing approach often used in the context of graph editing in node-link representations is to provide a palette of node and edge templates with default attribute values. The user simply chooses the template to be inserted. However, this is only available for single nodes and edges, and it might be necessary to update default attribute values afterwards. A third existing approach combining specification and insertion of nodes and edges utilizes gestural interaction. Examples can be found in [FHD10]. To reduce the user's workload, it would be worth to further investigate techniques supporting a seamless combination of specification and insertion of single as well as multiple nodes and edges.

3.1.2. *Apply Edit Operation*

Tasks belonging to this category are important to achieve a data change. According to existing *data manipulation languages*, all elementary data characteristics of a graph can be edited by applying any of three edit operations. If new nodes or edges have to be added to an empty or non-empty graph, the *insert* operation must be applied. If existing nodes or edges have to be removed, the *delete* operation must be utilized. If specific attribute values associated with existing nodes or edges have to be changed, the *update* operation must be used on them. This leads to the following tasks:

- Apply the insert operation to add a specified set of nodes and/or edges to a graph (edit the graph's structure).
- Apply the delete operation to remove a selected set of nodes and/or edges from a graph (edit the graph's structure).
- Apply the update operation to change a selected set of node and edge attribute values (edit the graph's attribute values).

Note that a set of nodes, edges or attribute values may contain a single, multiple or no elements. Applying an edit operation to an empty set results in no data change. Applying update operations to a set of attribute values always includes defining their target values.

It is important to mention that the set of edit operations proposed in this thesis differs from those in [Bau06]. The major reason for doing so is to support the editing of nodes, edges or attribute values in a direct rather than an indirect way. BAUDEL'S edit operations are: change an attribute value of the selected object(s), clone the selected object(s), delete the selected object(s) and add or remove attributes from the selected object(s). Hence, these edit operations do not allow to add new nodes or edges directly. Instead a user has to clone existing nodes or edges and change their attribute values afterwards, which is cumbersome. Moreover, this indirect strategy assumes that a set of nodes and edges already exists in the graph, which is not always the case. The creation of graphs from scratch is impossible this way. By using *insert*, *delete* and *update* operations, the graph's structure and the graph's attributes can be directly edited in any way. This also incorporates the creation of graph data. Thus, the edit operations proposed here are more general.

For applying certain edit operations, user input through interaction with the visual representation is needed. As reviewed in Section 2.3.2, existing techniques are based on direct manipulation [Shn83] and enable users to insert or delete single nodes and edges in node-link representations. Techniques for supporting these tasks in alternative visual graph representations as well as techniques for applying update operations on graph attribute data in general do not exist. **Hence, there is a need to develop novel techniques for supporting *Apply Edit Operation* tasks in various graph representations.**

3.1.3. *Analyze Data*

The tasks of this category have the aim to obtain information or knowledge from the visual representation concerning the editing of the graph. One can distinguish between two general goals: i), analyzing the graph for determining erroneous or missing data subsets that need to be edited, and ii), analyzing the graph for obtaining knowledge concerning the effect of previously applied edit operations. An example where goal i) must be achieved is the manual maintenance of a co-authorship network where nodes depict authors and edges represent co-authorships. If new papers have been published, it might be necessary to add authors or co-authorships. To verify this, the network has to be analyzed concerning missing data. An example for goal ii) is the analysis of newly-established paths between nodes of a traffic network after edges have been inserted. Whether goal i), goal ii) or both have to be achieved depends on the specific use case. Moreover, the use case also influences whether such an analysis is confined to a local scope (e.g., a subgraph of interest) or must be performed globally. Changes to the graph structure almost always have local and global effects. An example is the insertion of edges. This increases the degree of certain nodes (local effect) but may also establish paths between nodes which might change the connectedness of the whole graph (global effect). Changes to the graph's attribute values are similar. On the one hand, an update operation leads to a data change of the selected attribute values (local effect). On the other hand, the value distribution of the whole graph is changed too (global effect).

For accomplishing graph analysis and achieving the mentioned goals, the user must perform graph analysis tasks. In this regard, task taxonomies exist in literature. A well known example is the task taxonomy provided in [Lee+06]. It was devised by considering existing task taxonomies for data analysis in general, by considering application examples of visual graph analysis and by taking tasks involved in user studies concerning graph visualization into account. LEE ET AL. divide tasks for graph analysis into four categories:

- Structure-based tasks: Tasks for analyzing adjacency, accessibility and connectedness of nodes. An example is *Find the set of nodes adjacent to a node* .
- Attribute-based tasks: Tasks for analyzing nodes and edges concerning their attribute values. An example is *Find a node with a specific attribute value* .
- Browsing tasks: Tasks for analyzing paths. An example is *Follow a given path* .
- Overview tasks: Tasks for estimating certain characteristics of the graph. An example is *Estimate the number of nodes of a graph* .

For further tasks within these categories, the reader is referred to [Lee+06]. The tasks of this taxonomy focus on analyzing nodes as entities of interest. Consequently, this taxonomy has been extended with tasks focusing on edges, highly connected subgraphs like cliques and subsets of nodes sharing similar attribute values [KPW14]. It has been shown, that the tasks of the original as well as the extended taxonomy can be decomposed into more fine-grained tasks such as *Identify node(s) or edge(s)* or *Retrieve attribute value(s) of a node(s) or edge(s).* An overview of such low-level analysis tasks is presented in [AES05] and [VPF06].

For performing *Analyze Data* tasks, *passive interaction* with the visual representation is necessary [Spe07]. During passive interaction the user does not take visible action, but gains information from a visual representation by just looking at it. This does not imply a static visualization.

A large variety of visual graph representations tailored to support these tasks have been developed. As reviewed in Section 2.2.2, node-link representations can support the identification of paths, matrices can facilitate the identification of cliques or non-existing groups of edges and implicit representations provide support for identifying a tree's hierarchy for example. Techniques for enhancing such representations have also been developed. Examples include labeling algorithms for better supporting the identification of nodes or edges [LSC08], techniques for highlighting nodes that have already been visited to support their relocation [ZK15; ZK14] and techniques facilitating comparison [TFJ12]. The determination of derived information from graphs e.g., the *clustering coefficient* [WS98] or the average of all values of an attribute, is often supported with individual representations [BHJ09]. Although a lot of approaches have been developed individually, they are rarely available in a single system or a software framework. Thus, there is a need for further work in this direction.

3.1.4. *Adjust Representation*

By performing *Adjust Representation* tasks the user aims to adjust a given representation so that it matches pending *Select or Specify Data to Edit-*, *Apply Edit Operation-*, and *Analyze Data* tasks. Thus, three different tasks can be distinguished in this category:

Adjust the visual representation according to ...

- *Select or Specify Data to Edit* tasks.
- *Apply Edit Operation* tasks.
- *Analyze Data* tasks.

To accomplish above listed tasks, a representation is necessary where the data subset of current interest is at least i) visualized and ii) visible to the user. If the current representation does not meet these requirements, it must be adjusted accordingly. For that reason, parameters of the visualization process must be changed. If i) is currently not met, the data of interest has to be visualized. For that purpose, it is necessary to adjust parameters of the filtering, mapping or both. Most graph analysis systems provide facilities for doing so. This includes settings dialogues for configuring the mapping and techniques for interactive filtering (also referred as brushing). If the current representation meets i) but not ii), parameters of the rendering must be adjusted. For that purpose, graph visualization systems are generally equipped with pan and zoom navigation techniques enabling users to selectively show or hide certain data subsets. By utilizing such techniques, it is up to the user which data subset is currently visible. Besides the adjustment of parameters having global effect on the representation, there is also the possibility to adjust the representation locally. An important concept in this respect are magic lenses [Bie+93]. Techniques utilizing this concept enable users to get an alternative view of the data in a local region of interest. The graph visualization system *CGV* for example, provides several lens techniques tailored to the analysis of graphs [TAS09].

Interaction techniques especially for accomplishing *Select or Specify Data to Edit* tasks are connected with *interaction costs* [Lam08]. An important aspect of these costs are physical motion times in order to point on graphical primitives. Thereby, shorter motion times produce less costs and are preferable. According to *Fitts' Law* [Fit54] it is valid, that the shorter the distance between graphical primitives and the larger their size, the shorter the time to point on them using a finger or a virtual pointing device. This means, such interaction costs depend on the visual representation. In turn, for the accomplishment of *Select or Specify Data to Edit* tasks, the representation could be adjusted so that these interaction costs are kept as low as possible. For doing so, the user is usually equipped with techniques for changing the area and distance of graphical primitives interactively. This includes global zoom in zoomable user interfaces (ZUIs), and local zoom in regions of interest with magnification lenses and fish-eye views. As most graph representations use 2D graphical primitives (except for edges), these techniques are applicable. However, increasing the size of graphical primitives decreases the representation's capacity to show data and hence also their effectiveness concerning *Analyze Data* tasks. For that reason, a trade-off between those aspects must be found by the user.

After selecting the data subset to be edited, it might be further necessary to adjust the representation so that techniques for applying edit operations can be utilized. As BAUDEL mentioned in [Bau06], different representations provide different ways to interactively apply edit operations. This statement is substantiated concerning node-link and matrix representations in the following Chapter 4.

For supporting data analysis, it is required to adjust the representation so that it becomes effective concerning the user's *Analyze Data* tasks. This includes tasks for discovering data subsets that need to be edited as well as tasks for analyzing the effect of a previously applied edit operations. Existing user studies verified, that node-link-, matrix and implicit representations, as well as varying layouts of these perform differently concerning specific tasks (see Section 2.2.2 and 2.2.3). However, by far not all *Analyze Data* tasks, variations of these representations and layouts have been considered. Moreover, the results of these studies are difficult to generalize as they are mostly based on a small number of test subjects and only apply within their individual settings. Thus, further research is needed to figure out whether, and in which degree a certain representation effectively supports which tasks. When adjusting the layout of a representation concerning specific analysis tasks, care has to be taken to preserve the user's mental map. This is especially true when editing larger graphs with established or constrained-based layouts. In such situations, standard layout algorithms are not applicable as they usually affect the representation globally and cannot consider application dependent layout constraints. Techniques for manually adjusting the node-link layout of a large graph cannot be used either because this would be simply too time-consuming. Consequently, when editing graphs with established or constrained-based node-link layouts, novel approaches supporting *Adjust Representation* tasks must be developed.

An important aspect for supporting the accomplishment of *Select or Specify Data to Edit*, *Apply Edit Operation* and *Analyze Data* tasks is to provide immediate visual feedback for the user's actions [Elm+11]. This also requires an adjustment of the representation, which is often performed automatically in context of the applied interaction techniques. Examples providing such automatic visual feedback are selection techniques that highlight the current selection immediately.

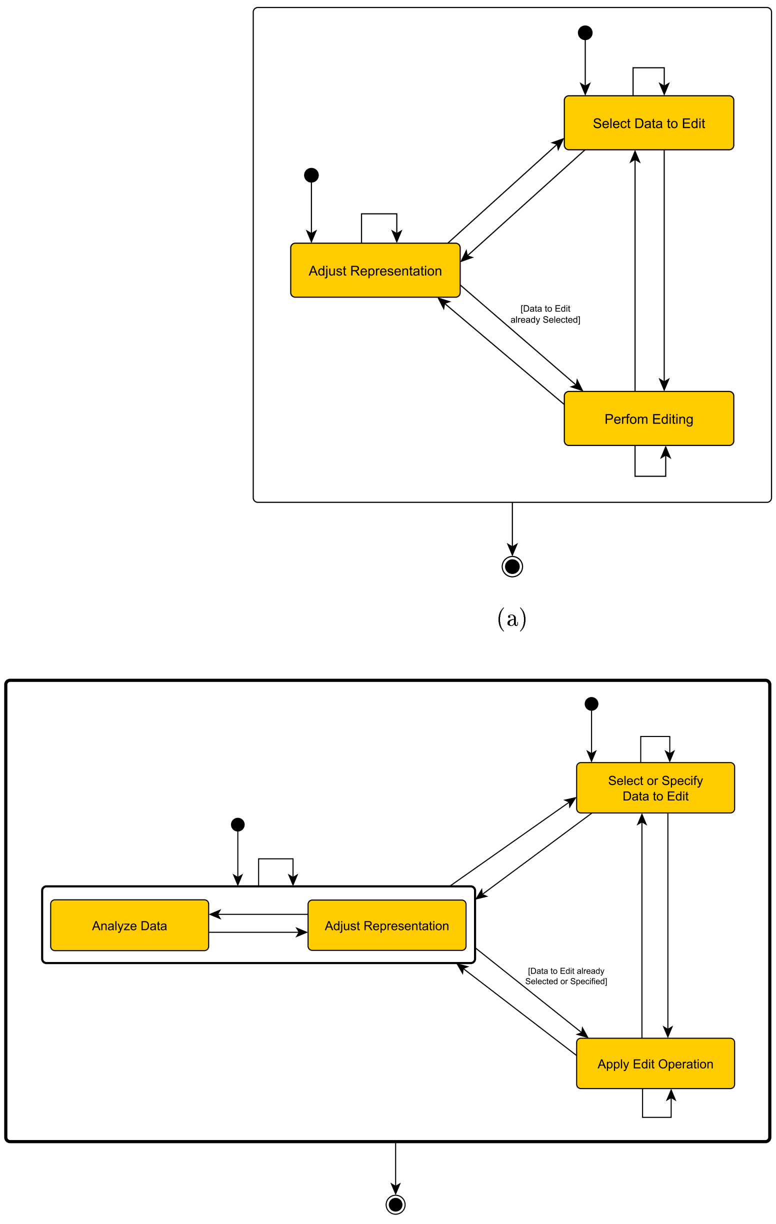
Except for providing visual feedback, adjusting a representation can also mean to switch to another view with a different representation. A lot of graph analysis systems provide multiple views with different representations side-by-side. The systems *CGV* [TAS09] and *Tulip* [Aub04] are examples.

Up to this point, tasks of visual graph editing and their current support were examined. However, for performing these tasks interactively, their interplay and processing order must be clear. This is addressed in the following section.

3.2. A Task-Transition Model for Visual Graph Editing

Visual graph editing is versatile. There are situations where the data to be edited is initially unknown and needs to be discovered first (a) and where the data to be edited is known beforehand (b). Moreover, one can distinguish scenarios where it is necessary to analyze the editing effect (c), or where it is not necessary (d), since only a missing value is added for example. For performing the tasks of visual graph editing according to these situations and for facilitating the development of novel techniques in this regard, it is necessary to examine the tasks interplay. This means, a model is necessary describing the task's processing order. Hereafter, the switch from one task, which has already been accomplished, to another task is called *task transition.* Because the task categories of visual graph editing contain a multitude of tasks, a model on the task level would become too complex and confusing. Thus, it is comprehensible to develop a model on the higher level of task categories providing an abstract view on tasks of visual editing and their transitions.

On the basis of the task categories described in the previous section and the *general editing model* from [Bau06], a task-transition model for visual graph editing tasks was developed in this thesis. This model encompasses but also extends the *general editing model* and thus is referred to as the *extended general editing model.* Figure 3.1 illustrates the *general editing model* and the *extended general editing model* side-by-side. An important addition is the *Analyze Data* task category. Now, the analytical aspect of visual graph editing as motivated before is included. Moreover, the interplay of data analysis and further tasks of the user becomes obvious. Transitions to all other task categories indicate the importance of the new *Analyze Data* category. To additionally encompass tasks for specifying data to be inserted (necessary for creating graphs from



(b)

Figure 3.1.: Activity diagrams of task-transition models for visual editing: (a) *general editing model* from [Bau06], (b) the *extended general editing model* including analytical activities of the user as well as the specification of data to be inserted.

scratch), the *Select Data to Edit* category was extended to *Select or Specify Data to Edit*. *Perform Editing* was renamed to the more precise *Apply Edit Operation* as described in the beginning of this chapter. In the following, the transitions of the *extended general editing model* are described in detail. Thereby, the scenarios (a) - (d) are considered and examples are given.

Start $\longrightarrow$ *Analyze Data / Adjust Representation* At the beginning it is assumed that a visual representation of a graph of interest is provided to the user. The user can start the editing session with *Analyze Data* or *Adjust Representation* tasks. If the user must discover the data to be edited first, the data must be analyzed (a). To make this possible, an adjustment of the representation can be necessary at the beginning. If the data to be edited is initially known (b), the user has to locate the data to be edited first. This includes obtaining information from the graph and is part of an analysis. In case the provided representation does not support these tasks sufficiently, *Adjust Representation* tasks might be accomplished. An example is the use of pan and zoom techniques at the beginning to be able to get an overview of the dataset [Shn96].

Start $\longrightarrow$ *Select or Specify Data to Edit* If it is known to the user that a graph is missing certain data (b), an editing session can start with specifying new data for insertion (e.g., when creating a new graph). If it is known that data needs to be updated or deleted, an editing session can also start with selecting that data. This does not necessarily include analyzing the graph before.

Analyze Data $\longleftrightarrow$ *Adjust Representation* The transition between *Analyze Data* and *Adjust Representation* appear frequently during the analytical activities of the user because the representation often has to be adjusted to sufficiently support the extraction of information. A typical example which can also appear in situations (a) and (b) is to adjust a representation in order to analyze details [Shn96].

Select or Specify Data to Edit $\longleftrightarrow$ *Analyze Data / Adjust Representation* A transition from *Select/Specify Data to Edit* to *Analyze Data* and *Adjust Representation* tasks is necessary if a user wants to continue analyzing the graph after selecting or specifying data to be edited ($\longrightarrow$). Examples for such a transition include the analysis of data after deselecting specific data that has already been edited, or the analysis of data for verifying a previously established selection. A transition in the other direction ($\longleftarrow$) often appears when the data to be edited is unknown (a). The user must localize data to be edited by performing *Analyze Data* and *Adjust Representation* tasks to eventually select them.

Select or Specify Data to Edit $\longleftrightarrow$ *Apply Edit Operation* After selecting or specifying data to edit, it is possible to apply an edit operation. An example for such a transition is to select a set of entities on which an update operation is applied afterwards ($\longrightarrow$).

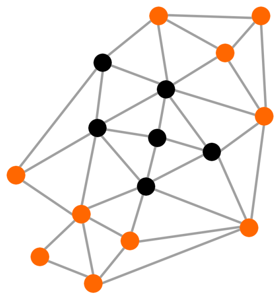


Figure 3.2.: Precise selection sometimes requires to accomplish multiple selection tasks. In this example, the orange highlighted nodes of a node-link representation cannot be selected at once, using a rectangular selection frame. If the selection frame supports unifying and subtracting individual selections, it can be used multiple times to achieve a selection of all orange nodes.

After applying edit operations, there exist situations where users have to perform *Select or Specify Data to Edit* tasks ($\longleftarrow$). This includes deselecting data that has already been edited or the refining a selection for further editing.

Analyze Data / Adjust Representation $\longleftrightarrow$ *Apply Edit Operation* The transition from *Analyze Data / Adjust Representation* to *Apply Edit Operation* tasks can only happen if a selection or specification of data to be edited already exists. This is illustrated with a *guard*[1] on the respective transition ($\longrightarrow$) in Figure 3.1(b). As already described, this could happen if a user selects specific data, continues analyzing this data in order to verify the selection and eventually applies edit operations.

After applying edit operations it can be necessary to switch to *Analyze Data / Adjust Representation* tasks for various reasons. Such a transition becomes necessary for finding further erroneous data (b, d), or for extracting information about the editing effect (c) for instance.

** $\longrightarrow$ End* At all times, the user can decide to finish the editing and analysis session.

Self loops *Apply Edit Operation*: It can be necessary to apply edit operations subsequently, for instance when an update operation with wrong parameters was applied and the results need to be corrected. Moreover, it is possible to delete data immediately after they have been inserted by mistake.

Select or Specify Data to Edit: Sometimes *Selection* tasks have to be accomplished several times for precisely selecting a subset of the data or for correcting a selection. This generally depends on the selection techniques and the visual representation at hand. Figure 3.2 illustrates an example where the data subset to be selected (highlighted in

[1]A *guard* in UML is a condition that must be true in order to traverse a transition.

orange) cannot be selected at once using a selection frame. If the selection frame supports unifying and subtracting individual selections, it must be used multiple times to achieve the required selection.

Analyze Data / Adjust Representation: The extraction of information from the data can usually be performed repeatedly with a given visual representation. As described in [KPW14], a sequence of obtained information may constitute the need for further analysis. Usually, this depends on the user and her goals. For recognizing erroneous data for instance, it is often necessary to extract, relate and compare information several times (b). A second example is the inference of information concerning the effect of a previously performed editing operation (c). This usually requires to perform multiple *Analyze Data* tasks.

Often it is also necessary to adjust the visual representation subsequently. A classic example is the subsequent use of pan and zoom.

Non-existing transitions *Start* $\longrightarrow$ *Apply Edit Operation*: A transition from *Start* to *Apply Edit Operation* is not allowed as applying edit operations always require an existing data selection or specification, which is not provided at the beginning of an editing and analysis session.

Using the task categories and task transitions of the *extended general editing model*, various editing workflows can be described. Two examples concerning the scenarios (a) - (d) are provided next. Concrete workflows of real graph editing use cases are described in the following Chapters 4 and 5.

Exemplary workflow: Data to be edited is unknown (a) + analysis of editing effect is required (c) Consider a situation where the data to be edited is initially unknown (a) and where the editing effect must be further analyzed after applying an edit operation (c). The user starts with analyzing the initial graph representation to get an overview. By doing so, particular nodes with remarkable attribute values are discovered. To verify that these node attribute values are erroneous, the user adjusts the representation and extracts detailed information. Now it becomes obvious that these values are indeed erroneous. The user consults the data supplier and requests the correct values. After selecting the erroneous values, an interaction technique for updating them is utilized. Next, the representation is adjusted again so that the editing effect (e.g., concerning the attribute value distribution) can be analyzed. This analysis reveals that there is no further need for editing.

This workflow can be described as an instance of the *extended general editing model* as follows. The subsequent performance of tasks from the same category is denoted with *. *Analyze Data to Edit** $\longrightarrow$ *Adjust Representation** $\longrightarrow$ *Analyze Data to Edit** $\longrightarrow$ *Select or Specify Data to Edit* $\longrightarrow$ *Apply Edit Operation** $\longrightarrow$ *Adjust Representation* $\longrightarrow$ *Analyze Data to Edit**.

Exemplary workflow: Data to be edited is known (b) + analysis of editing effect is not required (d) Consider a situation where the data to be edited is initially known (b) and where the editing effect is not of interest (d). The user starts to analyze the initial graph representation for locating the data subset to be edited. As the current representation does not sufficiently support this task, it is adjusted. Afterwards, the user again analyzes the data in order to find the data subset of interest. This cylce is performed until the data subset of interest is found. Thereafter, the user selects the data of interest and eventually applies edit operations for correcting them.

This workflow can be described with the *extended general editing model* the following way: (*Analyze Data to Edit** $\longrightarrow$ *Adjust Representation*)* $\longrightarrow$ *Select or Specify Data to Edit* $\longrightarrow$ *Apply Edit Operation*.

3.3. Summary

In this chapter, a conceptual view on tasks of visual graph editing was provided. This chapter's aim was to examine the tasks to be performed by the user, in which order they have to be done and how they can be accomplished by using existing visualization and interaction techniques.

For describing the user's tasks, four important task categories concerning different goals were set up: *Select or Specify Data to Edit*, *Apply Edit Operation*, *Analyze Data* and *Adjust Representation*. These categories were described in detail along with their containing tasks.

The tasks processing order was modeled with an extension of the *general editing model* from [Bau06]. The novel *extended general editing model* depicts the interplay of all tasks involved in visual graph editing, now also containing tasks for specifying data to be inserted and analyzing the data. Thus, a comprehensive overview on tasks of visual graph editing is now available. This does not only aid designers in developing novel visualization and interaction techniques tailored to these tasks, it also allows to evaluate systems for visual graph editing with respect to the tasks to be performed.

As multivariate graphs consist of entities (nodes), relationships between entities (edges) and attributes associated with these (node attributes and edge attribute), they are a kind of multivariate data according to the definition by Ware [War04]. Thus, the task categories as well as the *extended general editing model* can be generalized to describe *visual data editing*.

For investigating how to extend graph analysis systems to support all tasks of visual graph editing, the tasks' support with existing visualization and interaction techniques was reviewed. While most of them are already well supported, some require further support. This mainly applies for *Apply Edit Operation* tasks but also concerns *Adjust Representation* tasks dedicated to established and constraint-based layouts of larger graphs. The following Chapters 4, 5 provide novel techniques addressing this issue. These techniques are described using a uniform structure.

4. Novel Techniques for Editing the Graph's Structure

As reviewed in Section 2.3.2, graph editing in visual representations can be beneficial but only a few approaches exist supporting the user in this regard. Such approaches are generally limited concerning two aspects: they only support the structural editing of graphs of moderate size and are exclusively based on node-link representations. The editing of graphs with hundreds of nodes and edges as well as editing based on matrix representations (see Section 2.2.2.2) have not been considered so far. For that reason, this chapter introduces two novel approaches for the visual editing of the structure of larger graphs. The approach presented in Section 4.1 is based on node-link representations and facilitates region-wise editing of graphs with established and customized layouts consisting of several hundreds of nodes and edges. Section 4.2 introduces an approach for inserting and deleting edges directly in matrix representations. Both approaches focus the *Apply Edit Operation* tasks as motivated in Section 3.1. The chapter closes with a summary in Section 4.3.

Contents

4.1. Semi-Automatic Editing of Graphs with Customized Layouts 47

4.1.1. Problem Analysis 47

4.1.2. Approach 48

4.1.3. Implementation 59

4.1.4. Use Case and Evaluation 59

4.1.5. Discussion 66

4.2. Using Matrix Representations for Graph Editing 67

4.2.1. Problem Analysis 67

4.2.2. Approach 69

4.2.3. Implementation 72

4.2.4. Use Case 72

4.2.5. Discussion 73

4.3. Summary 74

Major parts of this chapter appeared in the following two publications:

[Gla+14] S. Gladisch, H. Schumann, M. Ernst, G. Füllen, and C. Tominski. "Semi-Automatic Editing of Graphs with Customized Layouts". In: *Computer Graphics Forum* 33.3 (2014), pp. 381–390.

- The author of this thesis developed and implemented the main concept of this approach, carried out the user study and wrote major parts of this publication.

[Gla+15a] S. Gladisch, H. Schumann, M. Luboschik, and C. Tominski. *Toward Using Matrix Visualizations for Graph Editing*. Poster at IEEE Conference on Information Visualization (InfoVis). 2015.

- This approach is based on the master's thesis *"Touch Interaction for Exploring and Manipulating Graphs"* from Benno Waldhauer [Wal14] which the author of this thesis supervised substantially. The poster was written by the author of this thesis.

For this chapter, the published work was adjusted and extended in the following way:

- The wording was adjusted to match the wording of this thesis, and a figure was added.
- The relationships of the published work to the tasks (Section 3.1) as well as the *extended general editing model* (Section 3.2) are emphasized.
- A novel edge routing algorithm is presented (Section 4.1.2.4) which was developed and implemented in collaboration with Valerius Weigandt within the scope of a student project.

4.1. Semi-Automatic Editing of Graphs with Customized Layouts

Within the scope of this thesis, a novel approach for the structural editing of larger graphs with established and customized node-link layouts was developed. This approach focuses on supporting *Apply Edit Operation* tasks and *Adjust Representation* tasks concerning the graph layout. Before going into detail, the problems to be solved are analyzed.

4.1.1. Problem Analysis

As reviewed in Section 2.3.2, existing approaches to edit graphs visually rely on node-link representations. However, these approaches fall short of addressing larger graphs with hundreds of nodes and edges forming a customized and established layout.

Editing graphs with customized layouts requires maintaining the mental map that users have already developed of the data. This is especially true when the layout obeys certain application-dependent constraints, for example, when specific nodes may only appear in certain regions of the layout. Therefore, editing must not result in a global change of the graph layout, but the effects must remain local. Otherwise, the risk increases that the graph can no longer be recognized as the one that it had been before. Thus, care has to be taken to maintain the integrity of the mental map on incremental changes to the graph.

When editing graphs with customized layouts directly within node-link representations, users should be allowed to input their ideas or constraints of the layout into the system to control the final outcome. Placing single nodes or routing individual edges by hand results in local changes and obviously permits full customization of the graph layout. However, such manual editing becomes impractical when editing larger graphs: Positioning inserted nodes and routing each connected edge by hand becomes tedious work and would take a lot of time, especially when certain quality criteria (e.g., shortest overall length of all connected edges) have to be evaluated mentally.

On the other hand, one could use automatic graph layout algorithms after applying edit operations and compute high quality layouts with respect to general quality criteria. But these algorithms typically do not address customized layouts. Furthermore, most graph layout algorithms work globally which contradicts the need for local changes to preserve the mental map. A third difficulty is that computational costs rise significantly when considering multiple layout criteria for larger graphs, which hinders interactivity of edit operations.

In summary, fully manual editing techniques are not sufficient for larger graphs with customized layouts due to the immense effort required from the user, and editing techniques in combination with fully automatic layout algorithms dismiss custom application dependent layout constraints, result in global changes and lack mechanisms for controlling the desired editing outcome.

4.1.2. Approach

The novel approach to editing graphs is *semi-automatic*, combining the strengths of interaction (yet without intensive labor) and computation (yet with local and customizable effects to the graph layout). It addresses the editing of the graph's structure, more specifically, the insertion of new nodes and edges as well as the adjustment of the graph's constraint-based node-link layout according to these data changes. As constraints to the graph layout are application-dependent, hereafter, the following constraint is considered exemplarily: Nodes with a certain semantic may only appear in an associated region of the graph layout that represents this semantic. For developing a novel graph editing technique, one need to think about what tasks are to be interactive and what can be done automatically, and how visual feedback ties interaction and computation together.

User interaction Interaction gives the user control of what to edit (e.g., a node), how to edit (e.g., to insert) and of what the editing outcome will be in the graph layout, including the possibility for customization. However, the user shall not be burdened with placing nodes at exact positions or defining exact routes of edges. Therefore, in this approach *point-wise* interaction is relaxed to *region-oriented* interaction. Instead of specifying precise points, the user interactively defines a local region where an edit operation will take effect. This local region acts as a *coarse solution* for the layout of nodes and edges affected by the edit operations. Finding a *precise solution* is the task of automatic computations, effectively reducing the user's workload.

Automatic computation Algorithmic calculations enhance the interactive editing by providing suggestions for exact node positions and edge routes. The underlying algorithms consider a set of layout criteria and existing constraints to compute appropriate results with respect to the local region as defined by the user. Different heuristic layout strategies leave room for customization by prioritizing specific layout criteria. The computational part relieves the user from evaluating the layout quality mentally and finding optimized solutions by hand.

Visual feedback The interplay of interaction and computation also demands suitable visual feedback. Of primary concern is to accentuate elements being considered for an edit operation and reduce visual emphasis of those that are not affected. Moreover, as the local region acts as a coarse solution and thus plays a special role, it must contrast to the rest of the graph layout. To ensure a smooth editing cycle while interactively exploring the space of appropriate layout solutions, visual feedback about the layout suggestions of the automatic computations is continuously provided to the user.

To support the aforementioned functionalities, a novel concept of an interactive lens for edit operations was developed – the *EditLens*. *Magic lenses* as introduced by BIER ET AL. [Bie+93] naturally lend themselves to the purposes described above. Such lenses are a visualization concept for providing alternative visual representations of the data underlying a local area of the screen on demand. The local region of the lens is defined by the lens' shape, size, orientation, and position. The actual effect is defined by a lens function. The lens functions used so far mostly alter the visual content within the lens. Examples include the magnification of a focus region or the clutter reduction of overcrowded parts of the display.

The *EditLens* proposed here has the function to edit the structure of the underlying graph and to refresh its node-link representation. According to the best of the author's knowledge, this is a novel type of lens function, not yet considered in the literature. With the *EditLens*, (i) interactive control by adjusting the lens, (ii) locally confined automatic computation of the lens function, and (iii) the presentation of corresponding visual feedback are incorporated in a single editing tool.

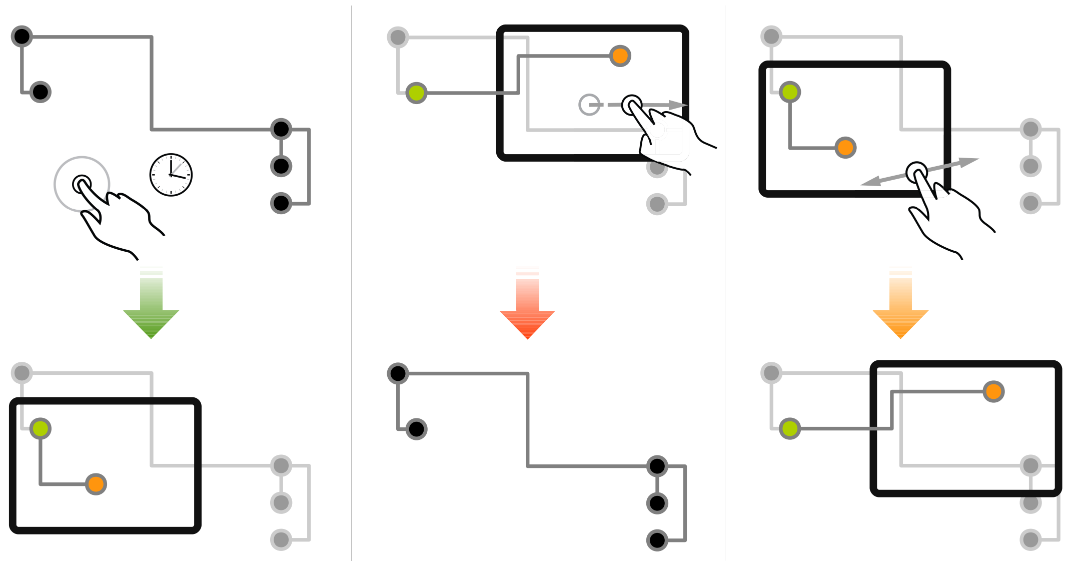
Without loss of generality, the *EditLens* has a rectangular shape which is suitable for most customized graph layouts. The rectangular lens can be resized (including adjustments to its aspect ratio) and repositioned interactively to define the local region where the edit operation is intended to affect the graph layout. By using a larger lens, the user widens the space of possible node positions and edge routes. Using a rather small lens can be useful when the user has a strong idea of a suitable solution in mind or to adhere layout constraints. As such, the *EditLens* utilizes the coarse-positioning skills of humans (as any magic lens does [Bie+94]).

Precise positioning is left to exact calculations by the computer. These calculations suggest appropriate node positions within the lens and suitable edge routes. The calculations are kept local in order to maintain the integrity of the already existing layout and to reduce computational costs. Naturally, computations with a local scope rarely lead to a global optimum. Therefore, the user is free to explore alternative solutions by adjusting the *EditLens* or prioritizing a different layout criterion. Eventually, the user may accept a suggested solution and commit the edit operation. Afterwards, there is still the option to manually refine a suggested solution via standard (pointwise) means of editing.

The *EditLens* has been designed to support the following tasks (see Figure 4.1):

- Insert nodes and edges into a graph and its corresponding node-link representation.
- Delete nodes and edges from a graph and its corresponding node-link representation.
- Refresh node positions and edge routes of a node-link representation.

The first two tasks are a combination of *Apply Edit Operation* and *Adjust Representation* tasks. The task to refresh node positions and edges routes addresses the layout of existing graph elements only and thus belongs to the *Adjust Representation* task category. The following Sections 4.1.2.1 - 4.1.2.3 describe how the *EditLens* supports these tasks. The insertion of a node with incident edges is described first, as it best illustrates how the *EditLens* works in general. Since inserting and refreshing edges both include



(a) Insert (b) Delete (c) Refresh

Figure 4.1.: Schematic depiction of insert, delete, and refresh with the *EditLens*. Nodes before and after editing are shown in black. The node being edited is highlighted in orange, whereas its adjacent nodes are represented in green. Nodes which are not connected to the node being edited are dimmed.

computing suggestions for edge routes, a suitable edge routing algorithm is presented in Section 4.1.2.4. Concrete gestures for interacting with the *EditLens* are provided in Section 4.1.2.5.

4.1.2.1. Inserting a Node

When inserting a new node to a graph and its node-link representation, usually the node's connections to existing nodes are inserted too. For this reason, the *EditLens* enables users to insert a node and its incident edges together. For doing so, the first step is to specify the node and its incident edges. This requires defining the node's identifier, its attribute values (if existent) as well as its neighbors. This can be accomplished via standard GUI controls. After loading the specified node and edges to the *EditLens*, layout suggestions for these elements, with respect to the local region defined by the lens, are presented automatically. For this purpose, two computational steps need to be carried out. First, an adequate node position must be computed and based on that, as the second step, the edges connected to the node must be routed. During this process, layout constraints and quality criteria have to be considered [Bat+98]. General goals are to avoid visual clutter and ambiguities, which usually means to use the available screen space efficiently.

As it is impossible to handle all possible criteria simultaneously [Pur02], a plausible set of four criteria is considered: In the first place, nodes should be easily discernible, which can be achieved by leaving sufficient space to other nodes. Second, neighbors should be easy to detect, which demands edges to be short. Third, edges should circumvent existing nodes, to prevent misinterpretation of connectivity. A well-accepted way to achieve this is to use orthogonal edge routing. Finally, edges should be easily traceable, which is the case for edges with as few bends as possible. This set of criteria is certainly not exhaustive, but it is assumed that alternatives can be taken into account on demand.

For finding an appropriate position p_v for the node v being inserted, the following three step procedure is suggested:

1. Determine a set $\mathfrak{R}$ of areas that are potential candidates for node placement. The areas must be within the *EditLens* and must be empty (free of existing nodes or edge segments).

2. Select a suitable area $R \in \mathfrak{R}$ within which p_v is to be computed. Necessary preconditions are that R is large enough to contain the node's shape and that semantic regions of the graph layout are obeyed.

3. Compute an appropriate position p_v within R taking into account the layout criteria indicated before.

A sufficiently fast method to determine $\mathfrak{R}$ is the *dynamic space management* by Bell and Feiner [BF00]. Their method is based on *full* and *empty space rectangles* which describe occupied and empty space, respectively. The dynamic space management is initialized by creating full space rectangles for the bounding boxes of all existing nodes and edge segments. Note that the bounding boxes are enlarged by a certain δ to preserve a minimal gap between existing and newly placed graph elements. The dynamic space management can then be queried for *empty space rectangles* inside the *EditLens* to compile the set $\mathfrak{R}$ as illustrated in Figure 4.2.

The next steps are to select an $R \in \mathfrak{R}$ and to compute the exact position p_v in R. For these steps, the necessary conditions formulated in step (2) and the sufficient conditions as imposed by the addressed layout criteria must be considered. However, it is hard or even not possible to obtain node positions and edge routes that fulfill all criteria in an optimal way. For example, orthogonal edge routes depend on the node's position. But the node position for the optimal route could be too close or even overlap with an existing node. To support the search for good positions, the *EditLens* can prioritize certain criteria. For this purpose, three placement strategies have been developed:

Node space first A node is easily discernible when there is enough free space around it. Following this thinking, R is selected as the largest empty space rectangle within the lens that falls into the semantic region associated with the node being inserted. Then p_v is computed as the center of R. Figure 4.2 illustrates the available empty space rectangles, with the largest one selected and the node positioned in its center. After placing the

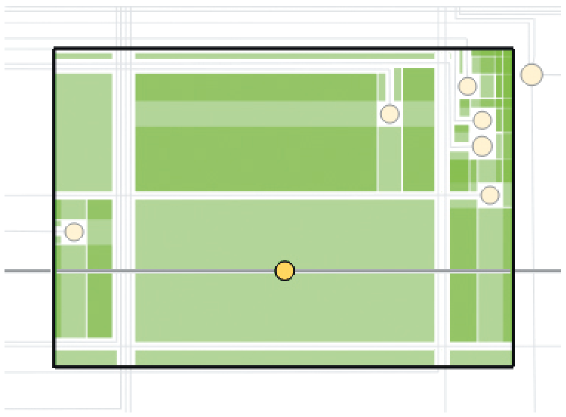


Figure 4.2.: Empty space rectangles (green) under the *EditLens* (dark frame) with the node being edited (yellow). Different shades of green indicate where rectangles overlap.

inserted node this way, the edge routing (see Section 4.1.2.4) is invoked to establish links to the node's neighbors.

Edge length first To identify neighbors of a graph easily, it makes sense to prioritize short edge lengths. According to this thinking, the sum of the lengths of the edges connected to the node being inserted should be minimized. As this sum depends on the node's position, the position within the *EditLens* where the sum of edge lengths is minimal must be found. However, the computation costs for edge routing prevent testing every possible position within the lens. Therefore, the following heuristic has been developed.

First, select R as the empty space rectangle that falls into the semantic region associated with the node being inserted and whose center c_R has the minimal sum of *Manhattan-distances* between c_R and every neighbor of v. Then an optimization problem is defined. Let (x, y) be a position in R, and $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be the positions of the n neighbors of v. Further, (l, t) and (r, b) denote the top-left and bottom bottom-right corner of R and d_v is the diameter of the shape of v. The goal to be optimized is:

$$\begin{aligned} f(x,y) &= \sum_{i=1}^{n} |x - x_i| + |y - y_i| \rightarrow min \\ \text{such that} & \\ x &> l + d_v/2 \\ x &< r - d_v/2 \\ y &> t + d_v/2 \\ y &< b - d_v/2 \end{aligned}$$

By solving this linear optimization problem with the simplex algorithm, an optimal position $p_v = (x, y)$ for v in R is derived. This heuristic, however, does not account for obstacles along the shortest routes (according to *Manhattan-distance*) to neighbors of v. The exact routes around obstacles are finally computed by an orthogonal edge routing algorithm, which is described in Section 4.1.2.4.

Number of edge bends first When tracing edges, bends along the way can be disruptive. Therefore, it makes sense to keep the number of edge bends to a minimum. Again the position at which edge routes have a minimal number of bends could be determined by invoking the edge routing for all possible positions and selecting the best one. As this is expensive, a simple heuristic to prioritize the reduction of edge bends was developed.

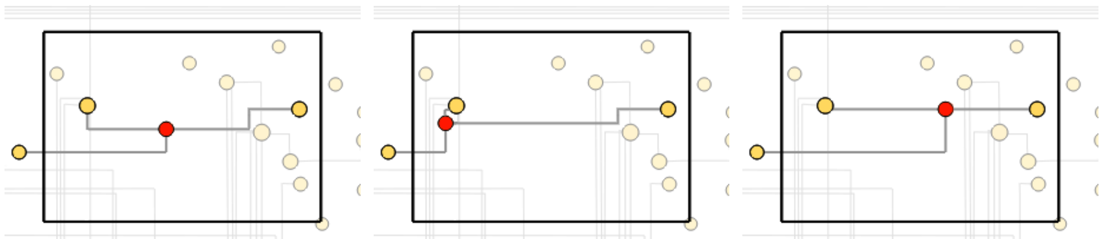
A key observation is that if the node v being inserted is aligned horizontally or vertically with one of its neighbors u, the edge (v, u) does not bend. Further, if two nodes u_1 and u_2 are already on a horizontal or vertical line, inserting v on that line in between u_1 and u_2 generates edges (v, u_1) and (v, u_2) that do not bend either. Therefore, the proposed heuristic searches exactly for such situations. Note, this heuristic does not consider obstacle nodes between v and its neighbors which could lead to edge bends when computing the final edge routes.

In a first step, it is determined whether straight horizontal and vertical lines between all neighbors of v exist (i.e., whether neighbors are aligned horizontally or vertically). Existing lines are then intersected with all empty space rectangles in $\mathfrak{R}$. Afterwards, an $R \in \mathfrak{R}$ is selected as follows.

If an empty space rectangle is intersected by both, a horizontal line and a vertical line, select that rectangle as R, and the intersection of both lines will be the position p_v of the node v. If no such rectangle exists, the heuristic looks for a rectangle R that is intersected by at least one horizontal or one vertical line, and place p_v on that line and centered in R. If multiple possible solutions are encountered during this search, one can either favor larger rectangles or shorter intersection lines to drive the final decision for R and p_v. After computing the node position, the edge routing algorithm is used to compute routes for connected edges.

Figure 4.3 compares the results of the three placement strategies. These strategies together with resizing and relocating the *EditLens* offer sufficient room for controlling and customizing the outcome of the node insertion. At the same time, the automatic calculations reduce the user's mental load and manual work.

If no suitable node position can be found by the automatic means due to insufficient space or due to unsatisfiable constraints (e.g., *EditLens* has been placed in a wrong semantic region), the color of the lens border changes to a signaling color such as red. The user can then either adjust the lens parameters or place the node manually, overriding any quality criteria. When a sufficient layout has been achieved, the insert operation can be committed by the user.



(a) Node space first (b) Edge length first (c) Number of edge bends first

Figure 4.3.: Three placement strategies determine a node's position based on prioritizing different layout criteria, including (a) favoring much free space around nodes, (b) shortening edges to neighbors, and (c) reducing the number of edge bends. The inserted node is shown in red, its neighbors are yellow, unaffected elements are dimmed.

4.1.2.2. Inserting an Edge

Similar to new nodes, new edges must be specified before they can be inserted into a graph. This requires selecting incident nodes, which can be accomplished either through GUI controls or by direct interaction with the nodes' visual representation. Moreover, if attributes are associated with edges of the graph, their attribute values must be set. For this purpose, GUI controls can be used again. After loading a specified edge (u, v) to the *EditLens*, the idea is to use the lens as a constraint for the edge routing, more precisely as a region through which the edge has to be routed automatically. Consider two nodes u and v dedicated to the same semantic region which have to be connected by an edge. It makes sense that the route of this edge will be within the confines of the dedicated region. However, when using a routing algorithm that is unaware of this region and simply calculates a route from the source u to the target node v, the result may not satisfy this specific constraint.

With the *EditLens*, the user first defines the local area within the dedicated semantic region of u and v through which the edge has to be routed. Second, the *EditLens* automatically calculates a single point p in the local region through which the edge has to be routed precisely. For determining p, the aforementioned placement strategies can be used again. Third, the automatic edge routing algorithm calculates a route from the node u to p and from p to the node v. This guarantees that the route will connect u and v passing through p. As the edge routing algorithm used in this approach aims to minimize edge length, it also indirectly aims to compute an edge route that remains within the bounding box of u, v, and p. Alternative to automatically calculating a point in the *EditLens*' region, the user can interactively specify the positions of a sequence of points within the lens through which the edge has to be routed. Afterwards, the edge routing will automatically calculate routes from u to v trough all the points of that sequence. This gives the user even more control of the editing outcome that will be suggested by the *EditLens* but requires more interaction effort. Again, by interactively changing the local area of the *EditLens*, using different placement strategies or defining different points in the *EditLens* the user can explore different editing results on-the-fly. When the user is satisfied with the layout suggestion, the insert operation can be committed interactively.

4.1.2.3. Deleting and Refreshing Nodes and Edges

In order to delete existing nodes or edges, or to refresh their positions or routes in the graph layout, they must be selected first. Such *Select* tasks (see Section 3.1.1) are supported with the *EditLens* too. Instead of burdening the user with precise selections, a coarse selection strategy is utilized following the idea of region-wise editing. When no entities are loaded to the lens, the selection of single nodes and edges closest to the lens center can be triggered interactively. The lens center is indicated by small cross hairs. A precondition is that entities to be selected are within the confines of the *EditLens*.

Delete operations can be realized easily. When a node is deleted, all connected edges are deleted as well. If the deleted node was an obstacle for other edges in the layout, the edge routes of these edges can now be improved by refreshing them. When deleting an edge, nodes that have been placed near the route of that edge might need further layout improvement to better utilize the screen space made available.

Refreshing the layout of a node or an edge can be reduced to virtual delete and insert operations in the following way. When refreshing an existing node, the node is deleted first and then re-inserted to a new position automatically. Analogously, when refreshing an edge in the layout, the edge is deleted first and then re-inserted with a new edge route.

4.1.2.4. Edge Routing

To compute suggestions for edge routes when inserting or refreshing edges, an edge routing algorithm is needed. According to the selected layout criteria concerning edges (see Section 4.1.2.1), such an algorithm has to meet two requirements: a) it computes orthogonal edge routes upon incremental changes to the graph which are rather short, have a low number of edge bends and circumvent existing nodes in the node-link layout. b) its runtime allows for interactive frame rates, so that users can explore different layout suggestions when moving, resizing, or switching the node placement strategy of the *EditLens*. Existing algorithms based on ad-hoc heuristics (e.g., the algorithm used in *Dia Diagram Editor 0.97*) meet b) but not a) as they may lead to aesthetically unpleasing edge routes, for instance overlaps with nodes. Thus, they cannot be used here. Existing algorithms for achieving high quality layouts (e.g., algorithm used in *yEd Graph Editor*) meet a) but not b) as they recompute the edge layout globally upon changes which is too time-consuming. Thus, they cannot be used either. A tradeoff between both aspects must be found.

Although the algorithm introduced by Wybrow et al. [WMS09] uses a global recomputation, it was a candidate as the authors claimed it would be able to "[...] calculate routings fast enough [...] during interaction". However, its applicability could not be

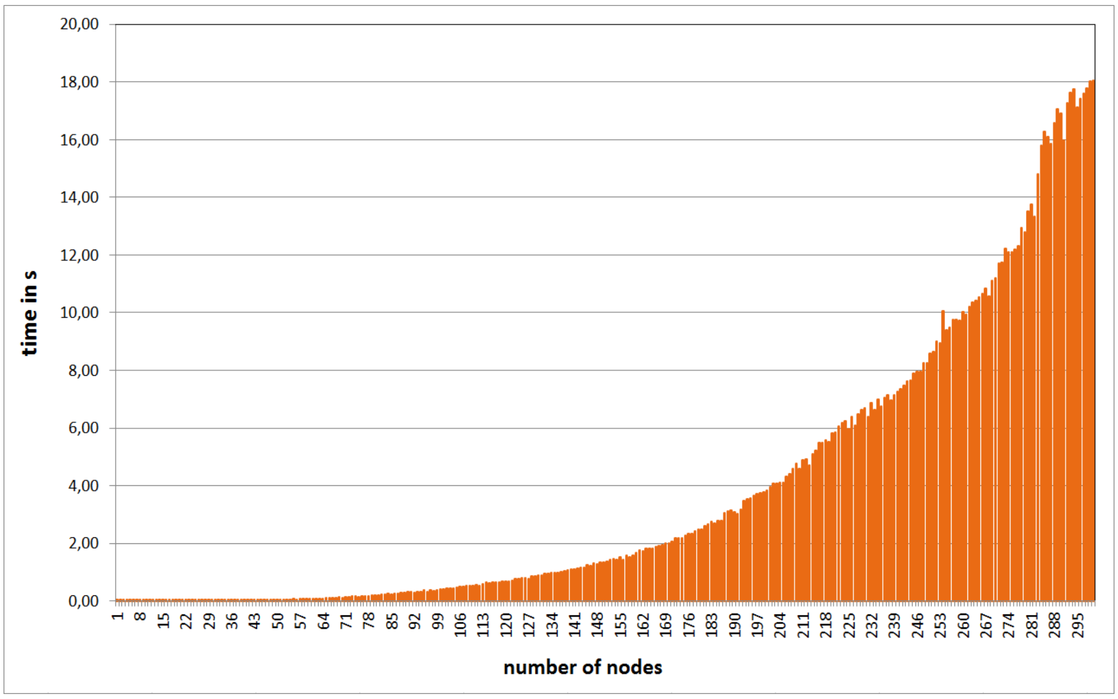


Figure 4.4.: Results of a performance test for computing orthogonal edge routes of a node-link layout with the algorithm from [WMS09]. The test graphs and respective node-link layouts were computed incrementally. The number of edges was selected to be two times the number of nodes.

verified with synthetical runtime tests. The test conditions were oriented towards the use case of the *EditLens* described in Section 4.1.4.1. A graph has been automatically created by incrementally inserting a new node with edges to two randomly selected existing nodes. Every new node was placed on a free position in the node-link layout randomly and the edge routing was computed using the original implementation of the algorithm from [WMS09]. The time needed for computing the edge routes of every insert operation was measured. Figure 4.4 shows the average results of multiple test runs. It is obvious that the time needed for computing the edge routes increases rapidly with increasing number of nodes. The computation of two edge routes in a node-link representation with 65 nodes and 130 edges took 100 ms on the test pc (Intel Core i7-3770k at 4.66 GHz, 16 GB RAM), which is already the upper limit for interactive frame rates. For 300 nodes and 600 edges, which is about the size of the graph for the *EditLens* use case, the algorithm took nearly 20 seconds. While the algorithm from WYBROW ET AL. might be able to compute edge routes in interactive frame rates for very small graphs, this test clearly shows its inapplicability for larger graphs. Hence, novel solutions are required.

Within this thesis, a novel orthogonal edge routing algorithm on the basis of the algorithm from [WMS09] was developed. Compliant with the node-insertion strategies of the *EditLens*, it requires nodes to be represented with overlap-free circles whose quadratic bounding boxes leave a certain distance between each other. Similar to the algorithm from Wybrow et al. and the algorithm used in [Roh+10], it consists of three steps which are summarized in the following. Details concerning these steps, achieved routing results and a comparison of the algorithm's runtime with the solution of Wybrow et al. are provided in Chapter 8.

Computation of potential edge routes The first step of the algorithm is to compute all potential edge routes of the node-link representation with an orthogonal visibility graph (OVG). This structure is based on the nodes of the node-link representation and hence must be updated upon every insert, delete or refresh operation on nodes. In contrast to [WMS09] where the OVG must be recomputed globally upon updates, the OVG used here can be updated locally. This reduces the overall runtime but comes at the cost of more memory usage.

Search of suitable edge routes Given a set of edges which must be routed, the OVG is used to search for orthogonal edge routes in the second step. Similar to [WMS09], the informed graph search algorithm *A-Star*[1] [HNR68] is used for this purpose.

Resolution of edge overlaps Edge routes provided by the previous step might overlap partially. The last step of this algorithm is to resolve such overlaps by shifting edge routes slightly. Wybrow et al. use a global, time-consuming strategy for this purpose. The strategy used in the new algorithm is local with respect to the newly routed edges and thus can be processed quickly.

4.1.2.5. Interaction Gestures

To take advantage of familiar mouse+keyboard interaction and the potential of novel multi-touch gestures, both modalities can be used to interact with the *EditLens*. This gives users the chance to choose the modality they deem best for the task at hand.

Standard GUI elements such as buttons, scroll bars, and menus are operated identically with mouse+keyboard and touch interaction. Interaction differs however, when editing in the node-link representation. The chosen gesture-design to enable user interaction has been inspired by previous work on touch interaction for graphs [Sch+10; FHD09]. Figure 4.5 summarizes the employed gestures. In the following it is indicated how users actually carry out *Adjust Representation-*, *Select Data to Edit* as well as *Apply Edit Operation* tasks.

Adjust Representation and Selection: To enable users view different parts of the graph in detail, the node-link representation is embedded into a zoomable space [Bed11]. Dedicated pan-navigation works via mouse drag in empty space or by two finger touch-slide

[1] *A-Star* is an informed search algorithm for graphs which is utilized to find the least-cost path between two given nodes of a graph. To enable a targeted and fast search, a heuristic cost function is used.

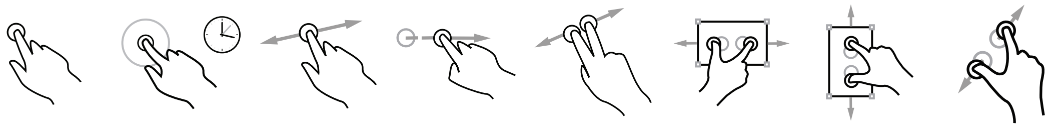


Figure 4.5.: Touch gestures used in the *EditLens* prototype.

in empty space. Zoom-navigation is activated via the mouse wheel or by using a pinch gesture, which is very common for this task. After spawning the lens with a long click or press in empty space, it can be used for selecting nodes or edges to be deleted or refreshed. A precondition is that no other elements are currently loaded to the lens. By applying a long click or long press on the lens, the selection of the node (with its incident edges) or edge nearest to the lens center (as described in Section 4.1.2.3) can be triggered. Individual nodes or edges can further be selected for examining their attribute values and for applying classic manual editing techniques. This can be done by single mouse clicks or by a single tap with a finger. For selecting multiple graph elements, the user can also use a lasso-selection gesture.

Apply Edit Operation: The lens is the main interaction element for applying edit operations and for adjusting the representation accordingly. It can be enlarged in x-direction, y-direction, or both combined by using pinch gestures. Resizing is also possible via the keyboard or the mouse. Repositioning of the lens is done via single finger slide or regular mouse drag. To move the lens over large distances, the user can use a double tap or double click as a shortcut, which triggers a smooth transition of the lens to the desired location. Adjusting the lens in any way initiates an automatic re-computation of the suggested position and routes of the elements being edited. When elements are about to be inserted or refreshed, users can accept provided layout suggestions by a single finger tap or mouse click on the lens. Elements loaded to the lens can be deleted using a flick gesture or by pressing the DEL key.

Fully manual editing of node positions and bends of edge routes is also supported as fall-back solutions. This can be accomplished through classic drag and drop gestures.

4.1.2.6. Approach Summary

The *EditLens* supports insert and delete edit operations on graphs as well as refresh operations on the graphs' customized node-link layouts. By interacting with the lens, users can explore different placement or routing suggestions for different regions and different layout strategies. This is considered as a key advantage of the *EditLens*. Such an exploration of layout solutions is currently not possible with either manual editing techniques or automatic layout algorithms alone.

Yet, as the *EditLens* computes heuristic-driven suggestions only, minor refinements might be necessary to fully satisfy the user. Such refinements of node positions or edge routes can be done using classic means of editing (e.g., dragging a node or the bends of an edge to slightly adjusted positions). As the suggested solution is already appropriate, the effort for manual refinement will remain at a reasonable level.

The interaction with the *EditLens* is supported by gestures of classic mouse+keyboard, as well as novel multi-touch interaction.

4.1.3. Implementation

A java-prototype based on the *MT4J* platform [LRW10] was implemented combining classic means for exploring graphs with the novel *EditLens* technique.

Graphs in GRAPHML or XGMML format can be shown as a node-link representation embedded into a zoomable space [Bed11]. Nodes are represented as circles whose size and color are set according to data attributes. Textual labels for nodes are placed with the help of a labeling algorithm [LSC08]. Edges are represented by orthogonal polylines. Exact attribute values of nodes and edges are presented upon selection in a dedicated interlinked table view.

The *EditLens* can be utilized to visually edit the graph as described above. The results of an editing session can be saved. To support the user in finding certain nodes of interest in large graphs fast (e.g., for checking their existence, refreshing or deleting them), an additional search functionality was implemented. If a node with an entered identifier exists in the graph, the view is smoothly translated to the position of this node and the node is highlighted.

With the proposed interaction techniques and the underlying algorithmic solutions for node positioning and edge routing, the implemented prototype is capable of dealing with graphs with several hundreds of nodes and edges. This is demonstrated by an application of the *EditLens* to a real-world problem from bio-informatics.

4.1.4. Use Case and Evaluation

To test the *EditLens*' usefulness and to gather user feedback, the proposed approach was applied to a real-world use case and a small user study was performed.

4.1.4.1. Use Case: Editing the *PluriNetWork*

A use case for the proposed approach is the manual maintenance of networks discovered in complex systems. An example is the *PluriNetWork* [Som+10], a literature-curated network of 365 nodes and 631 edges, where nodes represent genes and edges describe molecular interactions that are important for the cellular state of pluripotency in mouse cells. Bio-informatics scientists have developed the network from scratch using the editor of the *Cytoscape* platform [Sha+03]. The established layout follows a circuit-like design motivated by the analogy of the flow of gene regulation and the flow of electricity. As

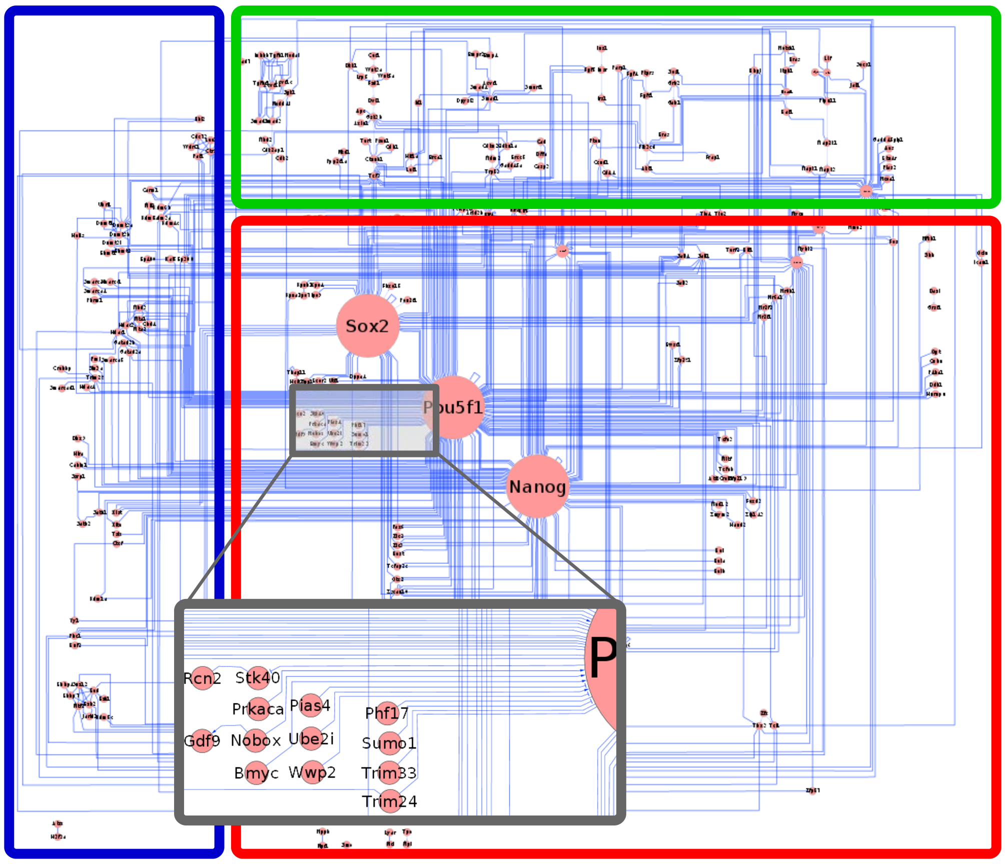


Figure 4.6.: The *PluriNetWork*. Marked are the three semantic regions (blue, green, and red). The detail view (gray) illustrates the difficulties of manual editing.

illustrated in Figure 4.6, the layout is divided into three distinct regions of semantically related nodes: signaling on the top (green), epigenetics on the left (blue), and transcriptional gene regulation in the center (red). These semantic information are only present in the node-link layout, not in the graph data itself. Thus, considering these regions is crucial when inserting new nodes to the graph or adjusting the layout of existing nodes. Once new verified knowledge becomes available in the literature, the network is edited manually. This manual editing is still carried out using the *Cytoscape* editor. However, with the increased size and complexity of the *PluriNetWork*, preserving the already existing semantic map is a major difficulty. The following description of the original editing procedure explains this.

Original editing procedure Given a newly reported interaction between two genes ($gene_1$, $gene_2$) as input, the manual editing workflow is as follows. First, it has to be determined if $gene_1$ and $gene_2$ are already present in the graph. If either of the genes does not exist, the curator has to specify the dedicated node first and insert it to the graph. After that, a spot for placing the node in the layout according to its associated semantic region must be found. If one gene already exists, the new node should be as close as possible to the existing node, but node overlap has to be avoided.

Then the curator specifies an edge for the newly reported interaction and inserts it to the graph. Next, a path from $gene_1$ to $gene_2$ is heuristically determined that (1) is short, (2) has a low number of bends, and (3) is as far away as possible from existing edges. Finding such a path could turn into a major challenge, as the optimal solution satisfying the three listed criteria is often not obvious or even does not exist. Often, a suboptimal solution was attempted, yielding a displeasing layout and triggering further follow-up adjustments of nodes and edges that were primary not affected.

If both genes already exist and only their interaction is new, the genes are usually already close in the network layout, because the layout reflects biological relatedness, and new interactions tend to connect related genes. In some cases, however, the nodes of $gene_1$ and/or $gene_2$ need to be moved to accommodate a visually more pleasing layout according to the mentioned criteria. Manually finding a path between $gene_1$ and $gene_2$ would then pose the same difficulties as described before. Further complications arise if a node has so many edges that there is insufficient space to add another one. In such cases, existing edges are rearranged, and if needed, the node is enlarged to attach the new edge to it.

The bio-informatics researchers transferred the graph and its node-link representation to the *WikiPathways* platform [Kel+12] to enable community contributions. However, the built-in editor of *WikiPathways* turned out to be cumbersome, effectively inhibiting any public editing of the *PluriNetWork*.

The original editing procedure can be described with the *extended general editing model* from Section 3.2. For simplicity, this is demonstrated exemplarily for the case where $gene_1$ and $gene_2$ are not present in the graph. The task sequence to be accomplished is summarized in the following. To express this task sequence with the *extended general editing model*, individual tasks are assigned to the task categories of visual graph editing as described in Section 3.1.

1) *Determine if nodes for $gene_1$ and $gene_2$ exist in the graph:* This task usually includes analyzing different parts of the node-link representation and thus can be decomposed to *Analyze Data* and *Adjust Representation* tasks.

2) *Specify node for $gene_1$:* This task belongs to *Select or Specify Data to Edit*.

3) *Insert node for $gene_1$:* This task can be interpreted as *Apply the insert operation to add a node* and hence belongs to *Apply Edit Operation* tasks.

4) *Determine a suitable node position for $gene_1$ in the node-link layout:* The determination of a suitable node position usually includes obtaining information from different parts of the node-link layout. This usually requires to accomplish *Analyze Data* and *Adjust Representation* tasks.

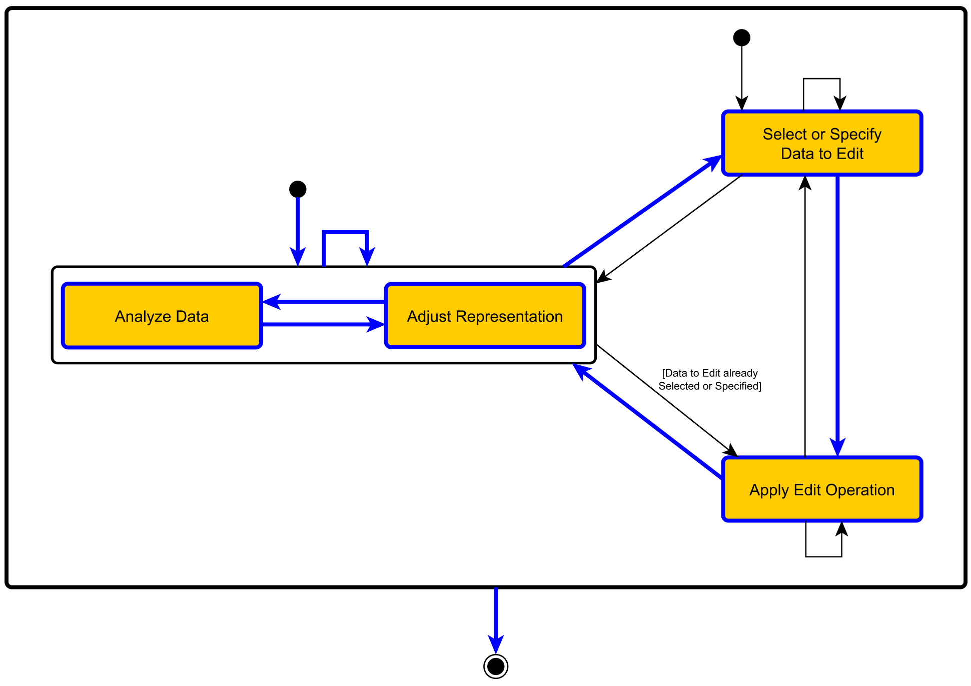

Select or Specify Data to Edit
Analyze Data
Adjust Representation
[Data to Edit already Selected or Specified]
Apply Edit Operation

Figure 4.7.: Using the *extended general editing model* to describe the original procedure for inserting two genes and their gene interaction in the *PluriNetwork*. Task categories and essential task transitions of this editing procedure are highlighted in blue.

5) Specify node for $gene_2$: This task belongs to *Select or Specify Data to Edit* tasks.

6) Insert node for $gene_2$: This task belongs to *Apply Edit Operation* tasks.

7) Determine a suitable node position for $gene_2$ in the node-link layout: This task includes *Analyze Data* and *Adjust Representation* tasks.

8) Specify edge between the nodes of $gene_1$ and $gene_2$: This task belongs to *Select or Specify Data to Edit* tasks.

9) Insert edge: This task belongs to *Apply Edit Operation* tasks.

10) Determine a suitable edge route in the node-link layout: This task includes *Analyze Data* and *Adjust Representation* tasks.

11) Adjust positions or routes of other nodes or edges to improve the layout: Adjusting the layout of elements in the representation belongs to *Adjust Representation* tasks.

As highlighted in blue in Figure 4.7, this task sequence is encompassed by the *extended general editing model*.

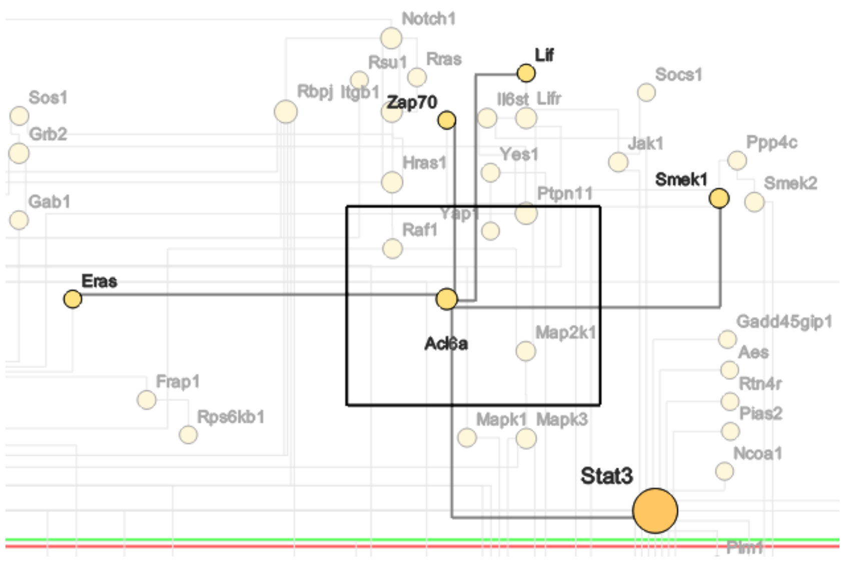


Figure 4.8.: *EditLens* utilized to insert the missing gene (node "Acl6a") and several missing gene interactions (edges). The green and red lines depict borders of semantic regions of the *PluriNetwork*.

Applying the *EditLens* To help the curators of the *PluriNetWork* in editing their data, the *EditLens* was integrated into the original editing workflow. The first step is still to classify a newly reported interaction of two genes ($gene_1, gene_2$) according to whether none, one, or both of the genes already exist in the layout. If none of the genes exist, then a node for the first gene is specified and loaded to the *EditLens*. Afterwards, the *EditLens*' insert operation is applied for the node dedicated to $gene_1$. Then $gene_2$ and its connection to $gene_1$ are specified and inserted together. If one gene is already present in the layout, only the latter tasks must be carried out. Figure 4.8 shows a screenshot of the *EditLens* used to insert a missing gene and multiple missing gene interactions. If both genes already exist, the *EditLens* is applied to insert a new edge between $gene_1$ and $gene_2$. When genes need to be repositioned or gene interactions need to be rearranged for layout improvement in general, the *EditLens*' refresh functionality is employed.

Deletion of genes or interactions among them is not applicable in the *PluriNetWork* as all existing elements are backed by scientific literature. In the unlikely event that a scientific discovery should be refuted, the *EditLens* would support carrying out the necessary correction in the *PluriNetWork*.

When considering the case where both genes need to be inserted with the *EditLens*, the sequence of tasks to be accomplished by the user is greatly simplified compared to the original editing procedure. Due to automatic computation of the *EditLens*, determining exact node positions and edge routes by hand (step 4), 7), 10) of the original editing procedure) is relaxed to exploring and accepting suggested layout solutions. Moreover, as the *EditLens* is able to insert a node together with its incident edges, these tasks must not be carried out individually anymore (step 6), 9) of the original editing procedure). As both aspects considerably reduce the user's effort, the hypothesis was set up that integrating the *EditLens* into the *PluriNetWork*'s editing workflow could help the maintainers of the network in carrying out their editing tasks. A user evaluation has been set up to verify this.

4.1.4.2. Evaluation

For the evaluation, the aim was to gather qualitative feedback regarding the potential usefulness of the *EditLens*.

Pilot testing Prior to the user evaluation, a visualization specialist was recruited for pilot testing. This way, the feasibility of the study was confirmed. Moreover, the initial feedback of the pilot tester lead to minor improvements of the color coding of nodes and the line color of edges. Then a small qualitative study using a flexible interview format was performed.

Participants The primary interest was to receive feedback from domain experts who edit networks on a regular basis. Therefore, two of the researchers maintaining the *PluriNetWork* participated. In order to contrast expert feedback against feedback from more casual users, two additional visualization experts were recruited, who had never worked with the *PluriNetWork* before. All participants were male and employees or students at a university. They were familiar with using interactive software, but none had used the *EditLens* prior to the study. The participants judged themselves as experienced in using touch gestures, which they knew from daily life using smart phones or tablets.

Tasks and devices For the first part of the study, the participants' task was to familiarize themselves with the prototype, especially with pan and zoom techniques, manual editing techniques, and with the *EditLens*. The experimenter first demonstrated the available functionality and then the participants were prompted to try them on their own. The network presented throughout the study was an updated and improved version of the *PluriNetWork* as depicted in Figure 4.9. The introduction lasted about 15 minutes.

In the second part, the participants had to solve three tasks on the *PluriNetWork*. The first task was to insert a new node with its connections to existing nodes. For the second task, the participants were asked to insert a new edge between two existing nodes. Finally, the third task was to improve the layout of an existing node. All three tasks were motivated by real-world tasks from the context of the *PluriNetWork*.

The participants had to carry out all tasks using both, the original manual editing procedure and the novel *EditLens*. Manual editing was done with mouse+keyboard interaction, whereas the *EditLens* was operable via touch-gestures. Two participants (one domain expert and one visualization-expert) used the *EditLens* first, the other two participants used the manual editing procedure first. On every task, the participants

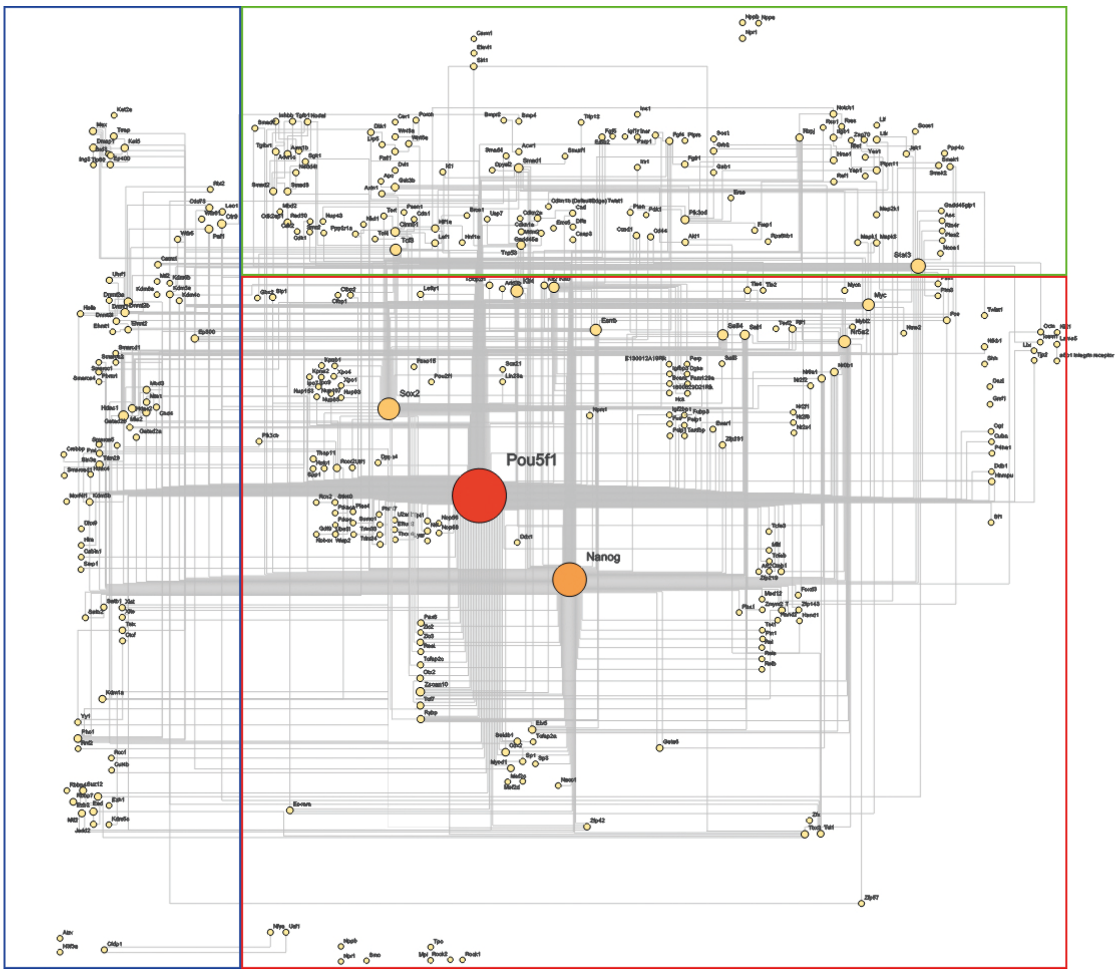


Figure 4.9.: The improved *PluriNetWork* as visualized by the prototype. Node size and color visualize node degree. Labels were placed using a dedicated labeling algorithm.

were instructed to think aloud meaning that they should give any feedback they have in mind. This second part of the study took between 30 and 40 minutes.

The study was performed using a regular desktop computer with a 23" touch-enabled monitor, which was arranged in a horizontal position. A reference sheet with the available multi-touch and mouse gestures as well as key mappings was presented on another monitor so that the participants could look them up quickly when needed.

Results All participants were able to finish all tasks using the original editing procedure and the *EditLens*. Questions regarding the usefulness of the *EditLens* were answered positively by all participants. The different layout strategies were also positively received by all participants. Ease of use of the *EditLens* was generally acknowledged. Concerning the question which technique they would prefer, both experts and both non-experts favored the *EditLens* in consensus with the hypothesis. One domain expert said: "the *EditLens* is very useful and can clearly reduce the editing effort". A visualization expert said: "the automatic suggestion of node positions and edge routes is obviously beneficial".

Additional comments suggest that there is room for improving the proposed approach even further. One participant of the study suggested routing inserted edges along existing edges to create bundles of edges. Another user would have liked to experiment with alternative lens shapes. Note that this study did not formally control all influencing factors (e.g., mouse+keyboard vs. multi-touch, horizontal vs. vertical display, simple vs. complex editing tasks). Therefore, the results obtained are to be understood as qualitative indicators to be confirmed in future studies.

4.1.5. Discussion

A semi-automatic approach to editing graphs with customized layouts was presented. The novel *EditLens* combines interactive means with automatic computation. In contrast to existing point-wise editing techniques, the *EditLens* follows a region-oriented paradigm. This greatly eases performing *Apply Edit Operation* and *Adjust Representation* tasks, as the user only needs to define a coarse region where an edit operation affects the layout of the representation. The algorithmic part of the *EditLens* automatically computes suggestions for precise node positions and edge routes, which the user can customize on-the-fly by adjusting the lens or selecting different placement strategies. Such a lens has not been considered in the literature yet.

A prototype implementation of the proposed approach has been applied to a real-world problem in the context of manually curating a larger biomedical network. It was demonstrated that the editing procedure used by the curators of the network can be expressed with the *extended general editing model* from Section 3.2.

In an initial user evaluation the utility and usefulness of the *EditLens* was tested. Overall the *EditLens* received positive feedback from domain experts as well as casual users making it a valuable alternative to the existing tools for editing data through interactive visual means.

An important point for discussion is the scalability of the *EditLens*, in terms of both computation and the number of edited graph elements. The proposed solution works with the hundreds of nodes and edges of the *PluriNetWork*. As the computational costs are usually bounded by the size of the *EditLens* and the limited number of entities affected by an edit operation, it can be expected that even larger graphs are possible to work with. Critical are refresh operations that affect high-degree nodes with many edges. In this case, the fast edge routing algorithm presented in Chapter 8 will reach its limits.

In terms of the number of edited graph elements, the focus was set on edit operations for single graph elements. Scaling to multi-element edit operations is a sensible next step. This requires considering additional constraints, for example maintaining intra-group ordering and alignment.

An interesting question to be investigated is the long-term effect of locally optimal layout solutions. It is yet to be observed, if the global layout quality deteriorates after many edit operations. The *PluriNetWork* application would be an ideal candidate to further study this behavior.

The *EditLens* has been designed for orthogonal graph layouts. To overcome this limitation and to be applicable more broadly, alternative aesthetic criteria (e.g., uniform edge-length, edges in bundles) and corresponding algorithmic solutions need to be considered. Lenses that automatically apply different constraints to different regions of a graph layout according to available meta information or data characteristics are an option. Investigating non-rectangular or even adaptive lens shapes is another interesting direction for future work. Balancing all influencing factors will be a challenge that requires further collaboration of interaction, visualization, and algorithm experts.

The proposed approach focuses on enabling users to achieve a data change. Although node-link representations (and existing exploration techniques for node link representations, e.g., the graph lenses presented in [TAS09]) provide support for *Analyze Data* tasks on their own, the support of such tasks has not been explicitly addressed by the *EditLens*. There are lots of possibilities for further assisting the user in this regard. Examples include the communication of parts of the graph's structure that might need to be edited (i.e. editing candidates such as unconnected nodes), as well as the communication of an edit operation's global effect to the data (i.e., the change of structural properties of the graph like its connectedness).

4.2. Using Matrix Representations for Graph Editing

The *EditLens* aims at better supporting the structural editing of graphs in node-link representations. Now, the focus is shifted to matrix representations, which have not been considered for visual graph editing in literature yet. In this section, a first approach for supporting the structural editing of graphs visualized as matrix representations is introduced. It enables users to insert and delete edges through direct interaction with the representation. However, in contrast to graph editing in node-link representations, the insertion of new edges through matrices does not require to compute an edge layout, because this is predefined by the structure of the representation.

4.2.1. Problem Analysis

The most common visualization approaches for graph exploration are node-link and matrix representations. As reviewed in Section 2.2.2, both have their individual advantages and support different analysis tasks. If a graph analysis involves tasks dedicated to both representations, node-link and matrix representations are ideally used in combination [HFM07].

What is widely accepted for graph analysis is not so clear for graph editing. Existing techniques for editing graphs visually are based on node-link representations exclusively. Matrix representations have not been considered so far, although it has been indicated that different visual representations have different strengths and weaknesses for editing tasks as well [Bau06]. Consequently, there are situations where the current support of specific editing tasks can be improved. An example is the editing of edges. Traditional techniques for edge insertion in node-link representations require interacting with the source and the target node, which can be time-consuming if these nodes are placed far apart in the layout. Edge deletion often requires picking the edge's line primitives beforehand, which can be difficult too. Moreover, as these techniques often address individual edges only, the user's effort is multiplied when multiple edges must be edited. This example motivates also considering matrix representations to support visual graph editing tasks.

To figure out which editing tasks can be sufficiently or better supported with matrix representations, these representations must be compared to node-link representations concerning certain requirements. As described in Section 3.1.4, to sufficiently support *Analyze Data* tasks, the representation must show the data effectively. To sufficiently support *Select or Specify Data to Edit* and *Apply Edit Operation* tasks, it is important that the represented data can be interacted with easily. This results in the following two requirements:

- The data must be presented effectively
- The data must be easy to interact with

Hereafter, node-link representations and matrix representations are examined concerning these requirements.

Node-Link Representations Assumed that a suitable layout can be computed, node-link representations offer an intuitive overview of sparse graphs and support the identification of paths. Dense graphs can lead to severe edge clutter, hampering overview and path-based tasks [GFC04] as well as the identification of existing and non-existing edges.

As node-link representations explicitly show nodes and edges, it is possible to interact with them. While interaction with the 2D node shapes is usually easy to accomplish, the interaction with edges can be difficult, because lines are not as easy to pick as shapes. Edge clutter aggravates this issue. This is especially relevant for touch interaction.

Matrix Representations When rows and columns of a matrix representation are ordered appropriately, matrices can provide a good structural overview of the data and allow users to find existing and non-existing edges quickly. Compared to node-link representations, edge crossings or overlaps cannot occur, and hence, edge visibility is guaranteed at all times and ambiguities are avoided. Moreover, matrix representations are particularly effective for analyzing dense graphs. However, they are ill-suited for path-based tasks [GFC04].

Concerning interaction, matrix representations have their strength for edge-based tasks. As a matrix cell corresponds one-to-one to an edge, edge insertion and deletion can be designed as interactions that are quite easy to carry out. Moreover, matrix cells are easier to pick than lines in node-link representations. On the other hand, nodes are only implicitly represented as rows and columns, which requires special treatment to interact with them for accomplishing node-based editing tasks.

In conclusion, node-link and matrix representations offer complementary ways to represent and interact with graphs. While node-link representations are rather suitable when the focus is on graph nodes, matrix representations have their strengths for edges.

4.2.2. Approach

The strengths of matrices for representing and interacting with the graph's edges shall be exploited to better support the visual editing of edges. Thus, techniques for supporting the insertion and deletion of the graph's edges in matrix representations have been developed in this thesis. These techniques are introduced next. Afterwards, it is described how node-link and matrix representations can be used in an interlinked-view setting to exploit the strengths of both representations for visual graph editing.

4.2.2.1. Edge Editing Using Matrix Representations

The aim of this approach is to improve the insertion and deletion of edges with novel techniques based on matrix representations. Different requirements have to be considered in this regard. First, interactive means must be provided to support the involved tasks. This comprises the interactive specification and selection of sets of edges as well as applying insert and delete operations. Second, the interactions used to accomplish these tasks should meet existing *interaction guidelines* [KPW14; Elm+11; Dix+97]. However, a difficulty is that there exist a multitude of such guidelines, which refer to different aspects of interaction and which are not free of conflicts. For that reason, a pragmatic approach for designing interactions is utilized here, where guidelines are prioritized that are important for graph editing:

- Interactions should be familiar to the user.
- Interactions should avoid editing by accident.
- Interactions should not conflict with other interactions commonly used to accomplish *Analyze Data* tasks.

Since applying insert/delete operations always requires to specify/select a set of edges first, the idea is to use single interactions for accomplishing both tasks seamlessly. This is similarly done by traditional editing techniques for node-link representations and aims to reduce the user's interaction effort.

To meet the first requirement, a check box metaphor is applied where matrix cells are used as primary targets for the interaction. The principle of check boxes is used widely in graphical user interfaces and users are familiar with it. When an edge exists, its cell is checked and filled with a color. When an edge does not exist, the corresponding cell is left

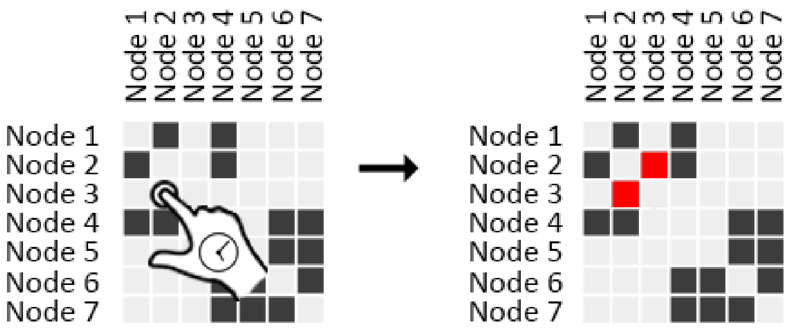


Figure 4.10.: Inserting a single undirected edge (highlighted in red) in a matrix representation using a long press gesture.

unchecked. Inserting and deleting edges can now be reduced to checking and unchecking matrix cells. When a cell is checked, the associated edge is inserted to the graph. When a cell is unchecked, the associated edge is deleted from the graph respectively. Checking or unchecking are typically triggered by a simple click or tap, which are quite short interactions. To avoid editing by accident and to be conflict-free with respect to interactions for analysis tasks, users are required to carry out a longer press on a matrix cell. The longer time that is needed to carry out the gesture also communicates the significant impact of the interaction, namely the change of the underlying data. Visual feedback for the progress of the long press is provided immediately. Figure 4.10 illustrates the interaction.

The introduced interactions can be used to insert or delete single edges. However, there are situations where multiple edges must be inserted. One could now use the technique described above multiple times, but then the editing effort is multiplied as well. For that reason, an alternative technique is used to edit multiple edges at once. In light of using familiar interactions, traditional techniques for multi-selection could be used to check or uncheck multiple cells at once. Examples include interaction with an additional modifier key (e.g., use SHIFT to un/check everything between two consecutive presses) or spatial selection via elastic rectangle or lasso. Yet, modifier keys are not always available (e.g., on tablets or tabletops), and the overhead of using a 2D spatial interaction is not really necessary. Therefore, the interaction used here for inserting or deleting multiple edges at once is designed as a simple long press followed by a 1D-restricted drag. The long press marks the first edge to be edited and the drag along all other cells (row-wise or column-wise) marks further edges to be affected (see Figure 4.11). This results in inserting or deleting all edges corresponding to the marked cells. An advantage of this technique is that it can be applied quite quickly. A drawback is that rows and columns must be ordered appropriately. As this is not always the case, the approach presented next includes a reordering mechanism.

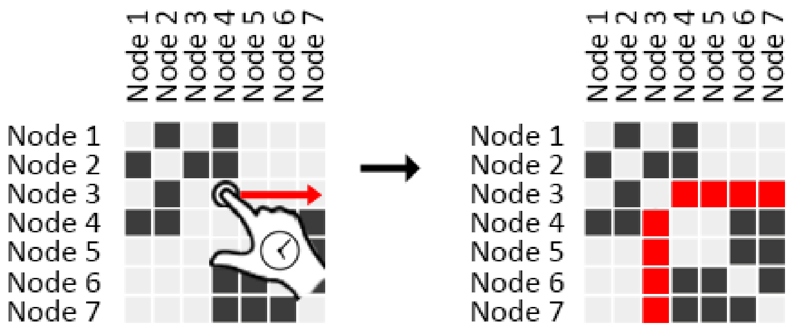


Figure 4.11.: Inserting multiple undirected edges (highlighted in red) using a combination of long press and drag gestures.

4.2.2.2. Structural Graph Editing Using Node-Link and Matrix Representations

With the proposed techniques for edge editing in matrices and the traditional techniques for node editing in node-link representations, benefits of both representations for the structural editing of graphs are exploited. To take full advantage of the editing techniques of both representations, node-link and matrix representations are now used in an interlinked view setting. As both representations visualize the same data, data changes become visible in both representations too. This way, the user can select the editing technique and representation that fits the task at hand best. However, this comes at the cost of reduced screen space for each representation. The interlinked view setting is further exploited to ease the accomplishment of the multi-edge editing technique in the matrix representation. When multiple nodes are lasso-selected in the node-link representation, the matrix rows and columns are automatically reordered so that the selected nodes are placed in the top left corner of the matrix. This allows to apply the multi-edge editing technique without reordering the matrix manually before. When deselecting nodes, the matrix ordering is reset to its original state. This is important to reduce interference with the user's mental map of the matrix.

The usefulness of this combined solution is illustrated with an example. Assuming a user is exploring a social network and the user's task is to follow paths between different communities. During the exploration using the node-link representation, the user recognizes a community where one person and its connections to all other persons in the community is missing. To insert this person's node, the user simply applies a double-tap on an empty position and thus utilizes an existing editing technique for node-link representations [FHD10]. To insert all missing connections, the user can now benefit from the interlinked matrix view. First, the whole community is lasso-selected in the node-link view and the matrix is reordered automatically. Inserting the connections is then as easy as applying the multi-edge technique to the appropriately reordered matrix. This way, the user is not burdened to apply traditional techniques for inserting single edges in the node-link representation multiple times. Because the matrix communicates the dense structure of the community well, the local effect of the editing is immediately

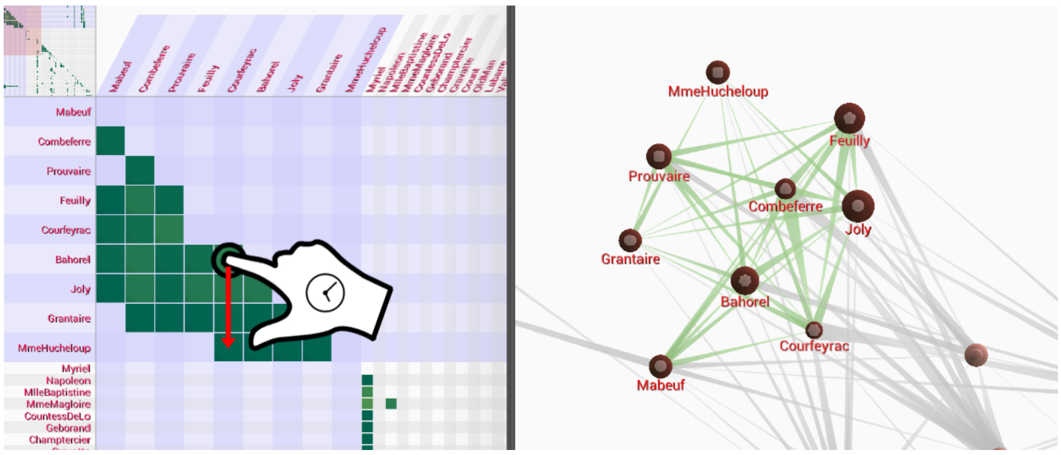


Figure 4.12.: Edge editing in an interlinked view setting. Highlighted nodes whose edges are about to be edited have been previously selected in the node-link representation. The edge editing is carried out in the matrix view. The underlying graph is directed.

visible. After confirming that the community is now connected correctly, the user resets the matrix ordering and continues the exploration.

4.2.3. Implementation

To demonstrate that the described combination of node-link and matrix representations can effectively support the editing of graphs, it was implemented by the supervised student Benno Waldhauer. The prototype application for an android tablet can be operated with touch-gestures. The proportion of the display area for each representation can be assigned interactively by moving a view separator with a drag gesture. Both representations support the typical pan and zoom navigation for exploration tasks. For this purpose, standard pinch and drag gestures are used. The node-link view is equipped with traditional editing techniques for inserting and deleting nodes [FHD10]. The matrix view implements the described new interaction techniques for edge editing and the automatic reordering upon selection in the node-link view. To circumvent common problems with touch precision when applying the multi-edge editing technique, the matrix cells dedicated to the currently selected nodes are additionally enlarged similar to a focus & context technique. This is illustrated in Figure 4.12, which shows the prototype during the deletion of multiple edges in the matrix view.

4.2.4. Use Case

The proposed approach can be applied to insert or delete edges of any graph. To demonstrate its applicability, it has been used to correct a character co-occurrence network from the novel *Les Misérables* [Knu09]. In this network, nodes represent characters and

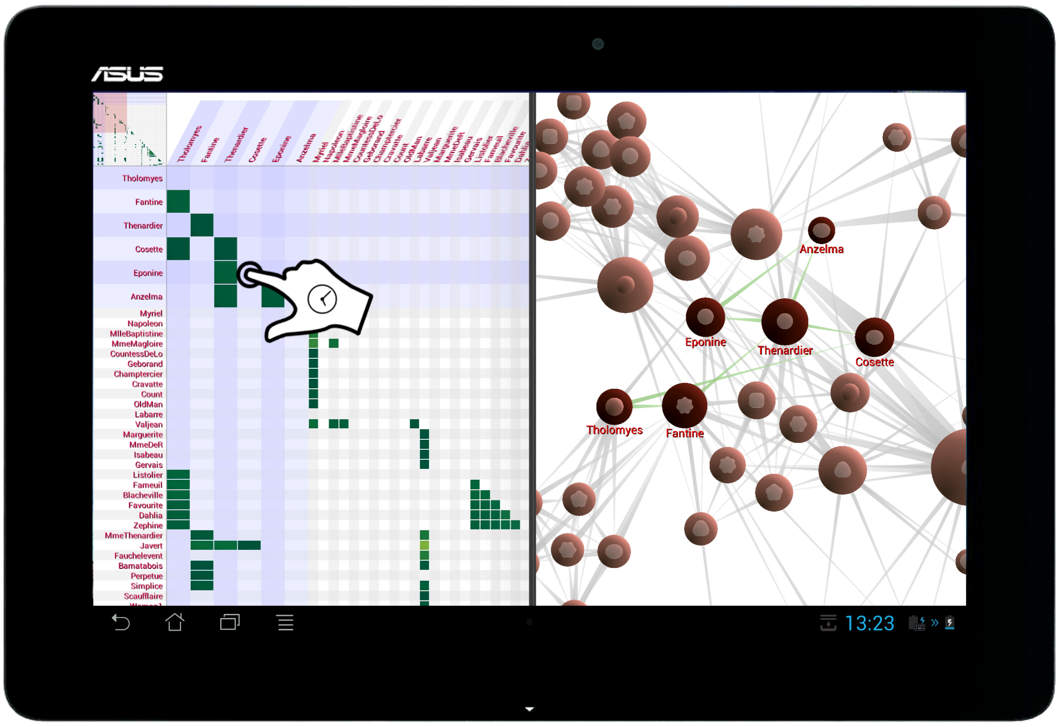


Figure 4.13.: Applying the matrix edge-editing techniques for inserting missing co-occurrences in a co-occurrence network of the novel *Les Misérables*. The colors of matrix cells as well as the thickness of edges in the node-link representation encode particular edge attributes. The node's size and symbol are used to encode specific node attributes.

edges depict whether characters occur in the same chapter. However, since this data set was manually prepared, it is subject to human error. As Donald E. Knuth published on his website [Knu15], there are several missing co-occurrences that still need to be corrected. This includes co-occurrences of the characters *Tholomyes*, *Fantine*, *Thenardier*, *Cosette*, *Eponine* and *Anzelma*.

To insert the missing co-occurrences, the suggested techniques were used. For that, the nodes of the characters with missing co-occurrences were selected in the node link representation which triggered the reordering of the matrix representation. Afterwards, as illustrated in Figure 4.13, the insertion of edges was carried out in the matrix. This procedure was performed repeatedly until all missing co-occurrences were inserted.

4.2.5. Discussion

In this section, matrix representations were investigated for graph editing. By comparing the characteristics of matrix and node-link representations according to requirements for visual graph editing tasks, it can be concluded that both complement each other in terms of representation and interaction. While matrix representations have their strengths concerning edge-based editing tasks, node-link representations can be advantageous for editing a graph's nodes. To improve the support of edge-based editing tasks, matrix interaction techniques were proposed. These techniques enable users to seamlessly perform specification and insertion, as well as selection and deletion of single as well as multiple edges by using a check box interaction metaphor. Thus, the support of *Apply Edit Operation* and *Select or Specify Data to Edit* tasks was focused. To exploit the benefits of node-link and matrix representations for the structural editing of graphs, an approach for using both representations in an interlinked view setting was presented.

Similar to *EditLens* from the previous section, this approach did not address the support of *Analyze Data* tasks beyond the abilities of matrix representations and dedicated exploration techniques known to science. To provide a more targeted support for such tasks, further research is needed.

The proposed techniques were used to correct a graph representing co-occurrences of characters in a novel. Although they worked well, a user study should be performed in future to quantify their effectiveness and to get user feedback. A comparison with traditional techniques for inserting and deleting edges in node-link representation would be interesting in this regard as well. Beyond that, matrix representations provide additional possibilities for graph editing. For instance, the editing of edge attributes using matrices could be considered. Moreover, a combination of specifying edge attribute values and inserting edges could be investigated to ease the user's workflow. Bimanual touch interaction might be suitable for this purpose. And last but not least, the benefits of interlinked node-link and matrix views for graph editing should be further exploited, for example by following a *NodeTrix* design [HFM07].

4.3. Summary

In this chapter two novel approaches supporting the structural editing of graphs were introduced. Both approaches aim to support the user's tasks beyond the possibilities of existing approaches for visual graph editing.

The first approach specifically addresses the editing of larger graphs with established and customized node-link layouts. It combines interactive means with automatic computation in a single tool called *EditLens* to better support *Apply Edit Operation* and *Adjust Representation* tasks. In contrast to existing point-wise editing techniques, the *EditLens* defines a local region where an edit operation affects an already customized node-link layout. Node positions within the lens and edge routes to connected nodes are calculated on-the-fly according to different aesthetic criteria and existing layout constraints. This reduces the user's effort, but still provides sufficient freedom to customize the edit outcome. Preliminary user tests have been conducted with researchers from bio-informatics who manually maintain a slowly, but constantly growing molecular network. As the user feedback indicates, the *EditLens* greatly improves their editing workflow.

The second approach addresses the structural editing of graphs in matrix representations, which have not been considered for this purpose before. Motivated by the results of a review of matrix and node-link representations, first techniques for graph editing using matrices were proposed. These techniques enable users to insert and delete single and multiple edges quickly by using a familiar interaction metaphor. They ease the editing of edges, which can be cumbersome when using traditional techniques for node-link representations. To exploit the individual benefits of node-link and matrix representations for the structural editing of graphs, an approach for using both representations in an interlinked view setting was presented. Although the usefulness and ease of use of the proposed techniques are indicated with a working prototype and an application to a co-occurrence network, user studies are still open.

While this chapter addressed the visual editing of the graph's structure, the following chapter focuses on the visual editing of attribute data associated with graphs. Since there are no techniques for this purpose yet, novel approaches in this regard are introduced.

5. Novel Techniques for Editing the Graph's Attribute Values

In this chapter, the focus is switched from editing the graph's structure to editing the attribute values associated with graphs. Since the *visual* editing of the graph's attribute values has not been considered in literature, the aim of this chapter is to provide first approaches in this respect. Thereby support for *Apply Edit Operation* tasks is focused as motivated in Section 3.1. However, in contrast to the proposed approaches for editing the graph's structure, the visual editing of attribute values further requires investigating support for *Analyze Data* tasks. The reason for this is that only few techniques already exist which support the user in this regard. For analyzing the graph's structure, users can resort to a large variety of existing graph visualization and interaction techniques.

From the perspective of the user, the common procedure of editing a graph's attribute values incorporates three steps: (i) identifying data to be edited, (ii) actually altering the data and (iii) verifying the editing outcome.

First, the attribute values to be edited must be determined. For initially unknown data, such values will usually be found in the course of a visual graph exploration. Therefore, the user has to carry out *Analyze Data* tasks as described in Section 3.1. A necessary requirement for supporting such tasks is the communication of attribute values. Especially values being potential candidates for editing, for example missing values or outliers, should be emphasized visually. As relations between attribute values and structural properties of the graph may exist, the graph structure should be visualized alongside the graph attributes. As visual graph representations usually cannot show all data aspects at once, interactive means to adjust the representation for the task at hand is required too. Consequently, a first question to be addressed is to select a suitable base visualization.

Second, the user must be enabled to edit data values. This includes selecting values to be edited (which belongs to *Select or Specify Data to Edit* tasks), defining new values from the attributes' value ranges, as well as committing the edit operations to the data (i.e., accomplish *Apply Edit Operation* tasks). A primary goal is to maintain the user's regular visualization (work)flow [Elm+11]. Hence, resorting to external tools is considered harmful. Instead, editing by means of direct interaction with the visual representation is pursued. Therefore, a second necessary requirement is that the visualization affords direct manipulation. Interaction techniques must be developed that keep the user's actual interaction effort low while still providing the necessary freedom for flexible data editing. As interaction precision is generally limited, a special challenge is the interactive definition of continuous attribute values.

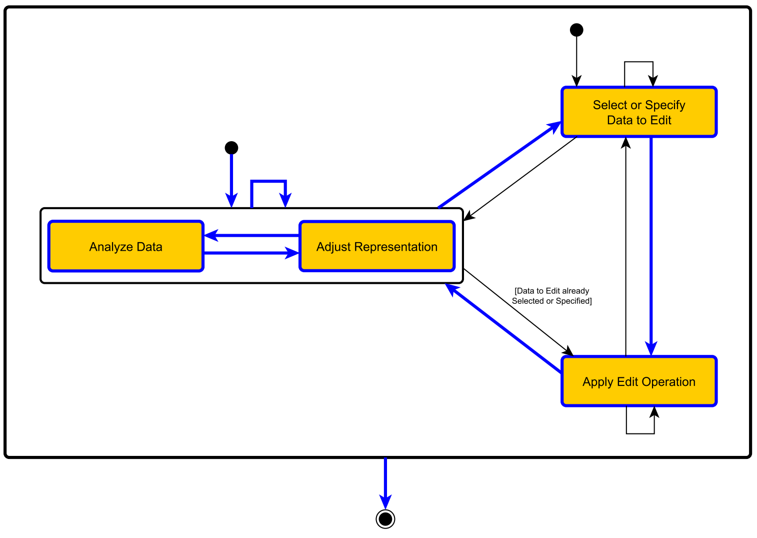

Select or Specify Data to Edit
Analyze Data
Adjust Representation
[Data to Edit already Selected or Specified]
Apply Edit Operation

Figure 5.1.: Using the *extended general editing model* to describe the task sequence commonly carried out when editing attribute values (highlighted in blue).

Third, after editing data values, it is typically necessary to verify the effect on the data by analyzing them. Therefore, informative visual feedback is a third key requirement. Visual feedback must be provided constantly when applying edit operations in order to communicate data changes. Most important in this regard is the immediate visualization of locally changed data values. Yet, an edit operation also takes effect more globally on the attribute's value distribution, which should be communicated as well.

From the perspective of the *extended general editing model* (see Section 3.2), this general attribute editing procedure is a sequence of *Analyze Data* and *Adjust Representation* tasks (identify and verify data) as well as *Select or Specify Data to Edit* and *Apply Edit Operation* tasks (alter data). Tasks and essential task transitions of this sequence are highlighted in blue in Figure 5.1.

In this chapter, two novel approaches for visually editing graph attribute values are introduced. The first approach described in Section 5.1 is based on node-link representations with *attribute-dependent layout* and enables users to edit node attribute values by directly moving nodes from one position in the graph layout to another. The second approach introduced in Section 5.2 is dedicated to node-link representations with *attribute-independent layout*. It follows the idea of integrating on-demand interfaces that offer controls for editing graph attribute values in place. Both approaches address the typical qualitative and quantitative scales of graph attributes.

Contents

5.1. Editing Node Attribute Values by Moving Nodes in the Graph Layout 81

5.1.1. Problem Analysis 81

5.1.2. Approach 82

5.1.3. Implementation 92

5.1.4. Use Case and Evaluation 92

5.1.5. Discussion 96

5.2. Integrating Radial Controls for Editing Graph Attribute Values 97

5.2.1. Problem Analysis 98

5.2.2. Approach 99

5.2.3. Implementation 107

5.2.4. Use Case and Evaluation 108

5.2.5. Discussion 111

5.3. Summary 112

The approach presented in Section 5.1 was submitted to *EuroVis* conference 2016.

- The author of this thesis developed the concept of this approach and is responsible for both use cases.
- The user study was carried out in cooperation with Christian Eichner.
- The publication was written in cooperation with Christian Tominski.

The approach presented in Section 5.2 is based on the poster:

[GT14] S. Gladisch and C. Tominski. *Toward Integrated Exploration and Manipulation of Data Attributes in Graphs*. Poster at IEEE Conference on Information Visualization (InfoVis). 2014.

- The author of this thesis developed and implemented the concept of this approach and carried out the user study. Additionally, major parts of this publication were written by him.

For Section 5.1, the work submitted to *EuroVis* 2016 has been adjusted and extended in the following way:

- Several detailed information were added.
- The problem analysis was replaced.
- Some figures were improved and added.
- The use case concerning the editing of document networks in Section 5.1.4.1 was reworked.
- The relationships to the tasks (Section 3.1) were established.

For Section 5.2, the poster publication [GT14] has been adjusted and extended in the following way:

- The concept was revised and extended.
- The prototypical implementation of the approach was additionally applied to a use case.
- Feedback was collected within a user study.
- The relationships to the tasks (Section 3.1) were established.

5.1. Editing Node Attribute Values by Moving Nodes in the Graph Layout

Within this thesis, a novel approach for visually editing node attribute values of a multivariate graph was developed. Its key idea is to use node-link representations with attribute-dependent layouts as a basis and incorporate dedicated editing interactions. The layout effectively visualizes the graph, where emphasis is put on characteristics pertaining to node attributes (e.g., value distribution or missing values). Attribute values are edited by dragging nodes in the node-link layout. Visual feedback constantly provides insight into the effect of the ongoing edit operation, enabling the user to experiment with different *what-if* scenarios. With this integrated approach, the cumbersome use of external tools for editing attribute data of graphs is no longer necessary.

Before describing details, general problems to be addressed when using node-link representations with attribute-dependent layouts as a basis for editing node attribute values are discussed first.

5.1.1. Problem Analysis

Node-link representations with attribute-dependent layout have proven to be useful for communicating node attribute values and the graph's structure simultaneously (see Section 2.2.3.1). Thus, they may serve as a basis for visually editing node attribute data. However, to support the three basic steps of the user's editing procedure, solutions for several problems must be developed.

For supporting the user in identifying data to be edited, a suitable attribute-dependent node-link layout must be found that reveals outliers and missing values as potential candidates for editing. As current attribute-dependent layouts are limited to visualize outliers only, they must be enhanced to additionally communicate missing values.

Using attribute-dependent layouts implies that the placement of nodes mostly depends on their attribute values. Thus, layout quality criteria for optimizing the communication of the graph structure can only be considered to a limited extent. Consequently, the layout might suffer from node overlaps or edge clutter. As these issues may hinder analysis, techniques for resolving them are needed.

Attribute-dependent node-link layouts have been developed for quantitative and qualitative node attributes. Generally, they can only represent a low number of attributes at the same time. If the user is interested in further node attributes of the graph, the representation must be adjusted. This however can lead to abrupt changes of the graph layout which makes it impossible to keep track of certain nodes. To avoid this negative effect, smooth transitions between changing graph layouts must be provided.

To enable interactive data changes on node attribute values, the node-link representation must be equipped with editing techniques based on direct manipulation [Shn83]. As the attribute-dependent layout implies that changes of node attribute values lead to changes of node positions, editing interactions based on spatially repositioning nodes in the graph layout must be developed. Thereby, interaction precision and interaction effort have to be considered.

In order to enable the user to carry out the last step of the editing procedure, i.e., to analyze and verify the edit outcome, visual feedback for communicating applied data changes must be provided. Most important in this regard is to represent the exact change of attribute values upon changing a node's position. Moreover, global changes to the overall value distribution must be communicated with suitable visual cues.

These requirements motivated the development of a novel visual editing approach.

5.1.2. Approach

The goal of this approach is to enable users to edit node attribute values directly in node-link representations with attribute-dependent layout. A key concept in this regard is direct manipulation [Shn83]. It is typically implemented as a spatial modification of on-screen objects. This idea of spatial modification is used as the core component of the proposed approach: Data editing is performed by spatial modification of graphical objects. Typical in visualization scenarios is that spatial modifications lead to adjustments of the visual representation. In this approach, however, spatial modifications manifest as actual changes of the underlying data.

As indicated above, a sophisticated editing solution needs three components, (i) an appropriate visualization, (ii) direct editing interaction and (iii) informative visual feedback. Depending on the data type of node attributes - quantitative or qualitative - the author proposes to employ different solutions for each component, which will be described next.

5.1.2.1. Editing Quantitative Values

Base visualization A sensible approach for visualizing quantitative values is to use a scatterplot layout of the graph nodes and an additional layer showing the graph's edges [Bez+10]. Figure 5.2 illustrates such an attribute-dependent representation of a fictional social network where nodes depict persons and edges represent friendships. The node positions encode two selected data attributes and node color visualizes the node degree (or a third selected attribute). This way, the structure and a subset of the attributes of the graph can be explored simultaneously and outlier values are immediately visible.

To additionally communicate nodes with missing values, a dedicated coordinate reserved for such nodes is defined on each axis. A reddish background signals that nodes with missing values are present in the data (see Figure 5.2).

In scatterplot layouts, nodes with similar attribute values lead to overplotting. This problem is tackled by showing cluster representatives instead of individual nodes. As illustrated in Figure 5.2, the representatives are shown as squares to make them easily distinguishable from regular nodes. The size of a square and its numeric label indicate the number of nodes contained in a cluster. On request, a cluster representative can fan

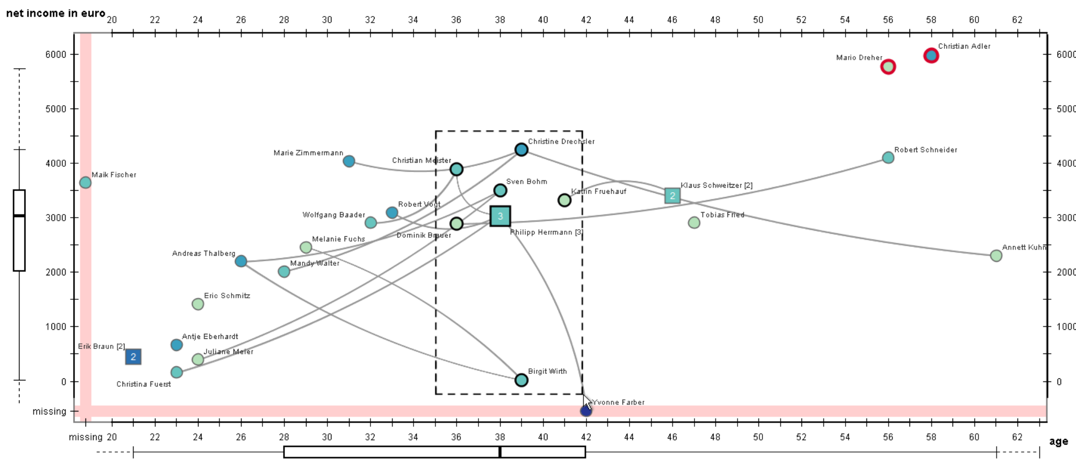


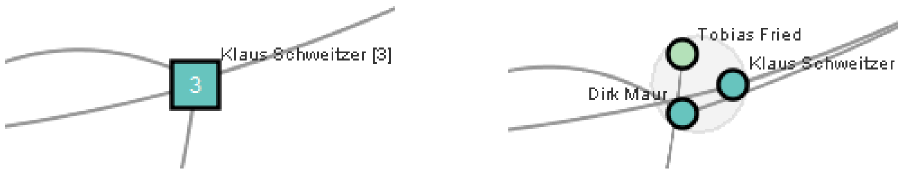
Figure 5.2.: Node-link representation with scatterplot layout. Nodes (fictional persons) are positioned according to two the quantitative attributes AGE and NET INCOME. Currently selected nodes are highlighted and their edges (friendships) are displayed. Nodes with missing values are placed separately on a dedicated coordinate along the axes.

out its contained nodes to enable their editing. Figure 5.3 shows a detailed example of a cluster representative and its fan out.

As the x and y coordinates of a node in the scatterplot layout are defined by its attribute values, graph layout algorithms are not applicable. Consequently, visualizing all edges at the same time often leads to edge clutter. To reduce edge clutter, edges are only shown for a selection of nodes of interest as described in [EW14].

To make layout changes comprehensible upon switching the node attributes represented by each axis, animation is applied as usual. Additionally, each node is labeled with its identifier string. For computing label positions, the labeling algorithm from [LSC08] is used. In this approach, the labeling algorithm is configured to reduce overlap among nodes and labels. An additional reduction of overlap among edges and labels is not pursued as it would increase computation time (which must be kept low for achieving interactive frame rates) and decrease space for placing labels significantly.

Direct interaction and visual feedback With the scatterplot visualization, a basis for interactive spatial modification of node positions for the purpose of editing is established. As common in interactive software, drag-and-drop gestures are employed to perform spatial modification. In other words, the user grabs a node and the position where the node is dropped defines the new attribute value(s) to be committed to the data. Yet in contrast to regular drag-and-drop, the user has to perform a long press before the dragging can start. The reasons are three-fold. Conflicts with any pre-existing interactions based on drag gestures are avoided, the risk of editing by accident is reduced, and the user is made aware of the fact that the underlying data will be manipulated. A



(a) Cluster representative (b) Cluster fan out

Figure 5.3.: Cluster representative and fan out.

similar initial long press has been used in both editing approaches described in the previous Chapter 4. While the drag gesture is being performed, the node position is constantly updated and a label indicates the exact value at the current position. An additional ruler indicates which values can be reached when the drag gesture is continued. The ruler helps the user to stay focused on the node being edited as it reduces eye movements toward the scales at the scatterplot axes.

Typically, editing is applied to a data value of a single node. Nevertheless this approach also supports the editing of multiple nodes, which have been interactively selected before. Thereby, two options are considered: absolute change and relative change. For absolute change, the attribute values of the nodes being edited are set according to the same new absolute cursor position. A node cluster will be the result. In contrast, relative change means that the attribute values are shifted according to the relative displacement of the cursor (with respect to the position where the drag gesture has begun).

With drag-and-drop the user can now freely set node attributes to any value being presented in the scatterplot layout. A general problem, however, is that values outside the current value range are not directly accessible in the scatterplot. Extra zoom and pan operations may become necessary to assign such extreme values. Yet, the consequences for this approach are limited as it is questionable if setting extreme outlier values is a regular editing task. More likely is the case that outlier values need to be corrected.

Low interaction costs are important to the ease of use of this editing approach. According to NORMAN, interaction costs can be attributed to the execution of an action and the interpretation of the result [Nor13]. Consequently, the basic solution described above is enhanced with additional tools and aids to further reduce the effort for editing data values and verifying the outcome.

In general, data editing relies on human users, who in this case perform spatial modifications of graphical objects. One issue is that the human motor system is accurate only within certain limits and prone to involuntary motion (e.g., hand tremor). As a consequence, exact editing requires concentration and focus, which in turn implies that the costs for carrying out editing tasks can be high. A possible solution to this issue is constraining the interaction, which is among NORMAN'S design principles for usability. In the context of this approach, the drag is dynamically restricted to the scatterplot axis of the attribute value the user is about to modify. The axis is chosen based on the maximal coordinate of the initial cursor displacement within a reasonable distance around the point where a drag gesture started. Once a decision for one axis has been made, the drag gesture remains constrained until it ends. The advantage of constraining the drag horizontally or vertically is clear: Involuntary manipulations of the second attribute are prevented. To enhance flexibility, although rarely necessary, a modifier key can be held down for unconstrained editing. This mode enables the user to modify node values with respect to both visualized attributes simultaneously.

To verify the result of an edit operation, the scatterplot visualization is constantly updated while editing. This way, the user can easily relate the edited value to values of other nodes, and outliers in the global value distribution are visible at first glance. Box plots along the scatterplot's axes are used to additionally indicate changes to the global value distribution. According to the definition of Tukey [FHI89], the solid whiskers of the box plots indicate the lowest and the highest existing attribute values in the lower and upper quartile which are not considered as outliers. For previewing outlier values, whiskers depicted in dashed lines are added showing the current threshold from which values are considered as outliers (see Figure 5.2).

What might be difficult to evaluate are outliers with respect to the local k-neighborhood of the node being edited (such local outliers do not have to be global outliers). However, such information can be important as there might exist correlations between connected nodes and their attribute values. Therefore, local outliers are also determined within the k-neighborhood of the node being edited if it contains a sufficiently large number of nodes for applying statistics. To increase the awareness of both, local and global outliers, respective nodes are highlighted with red outlines.

So far, outliers with respect to the two attributes visible in the scatterplot are considered. Extensions to detect and communicate multivariate outliers, as described in [DG92], can be implemented easily.

Increasing editing accuracy A key concern when inputting quantitative values via spatial modification is interaction precision. It is limited by the resolution of the input-output system and depends on the minimum-maximum value range mapped to the available pixels. The practical implication is that moving the cursor by one pixel results in a data modification by a constant delta. Particularly for continuous attributes with large value ranges, subtle edits can be difficult (sometimes even impossible) to perform via drag-and-drop gestures because the delta is too large. To allow for sufficiently small deltas, one can vary the minimum-maximum range by zooming. This, however, implies additional manual navigation and global changes of the state of the visualization.

For this reason, a light-weight focus+context fish-eye transformation is suggested, which is dynamically embedded into the representation. This transformation locally stretches the space for a focused interval of interest and compresses its context. By adjusting the transformation's magnification factor, the interaction precision can be increased gradually. Experimentally it could be determined that increasing the precision at fixed rates proportional to a base-10 logarithmic function is a suitable solution. As attributes usually have different value ranges, interactive means for adjusting the magnification factor independently for each axis are provided.

For precisely editing a node's attribute value by using the focus+context magnification, the user has to perform the following steps, as illustrated in Figure 5.4:

1. Long press the node to unlock it for editing.
2. Drag the node coarsely toward the target position.
3. Activate the focus+context transformation and adjust the magnification factor as necessary for the relevant axis.
4. Exploit the added precision to set the node's position exactly to the target value.

In step three, the focus+context transformation can be activated with an additional modifier key or by applying a gesture directly to the respective axis. In case of editing both values simultaneously, the user has to adjust the magnification factor of both axes if necessary. The area in the scatterplot which is magnified concerning both axes is highlighted with a solid black rectangle as exemplified in Figure 5.5.

In addition to improve interaction precision, the focus+context display can also be employed to resolve node occlusion in dense areas of the scatterplot. To this end, the focus interval can be coupled with the cursor movement in which case it automatically follows the user's exploration interest. To allow fast or precise positioning, the focus can be moved at focus or context precision. For that, the mode switching strategy described in [ACP10] is used.

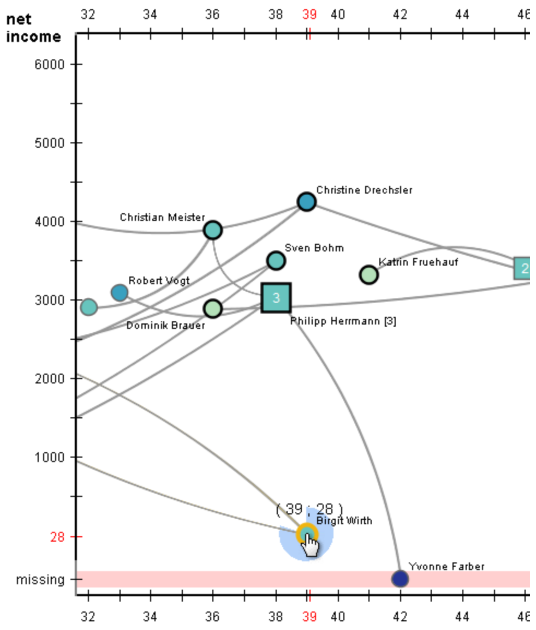
Although being generally useful, the focus+context approach is not able to handle all possible border cases. For worst-case scenarios it is still necessary to resort to alphanumeric keyboard input.

The approach described so far focuses on the editing of quantitative values. It could also be employed to edit qualitative values, provided they are faithfully represented in the scatterplot (e.g., by following the approach presented in [Ros+04]). But still the specific character of qualitative data calls for a dedicated strategy for editing them.

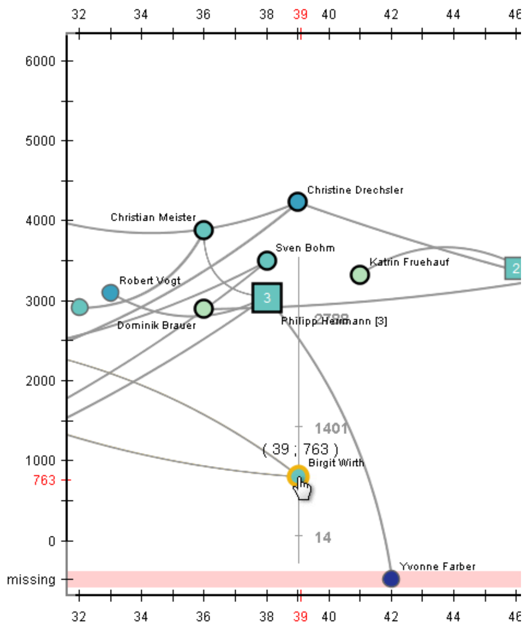
5.1.2.2. Editing Qualitative Values

Base visualization In general, visualizing qualitative data in a scatterplot leads to severe over-plotting at exactly the positions associated with individual values. Moreover, when mapping qualitative values to axes, a distance between values is communicated where no distance exists. Therefore, a scatterplot is not a good choice as a base visualization for communicating qualitative data.

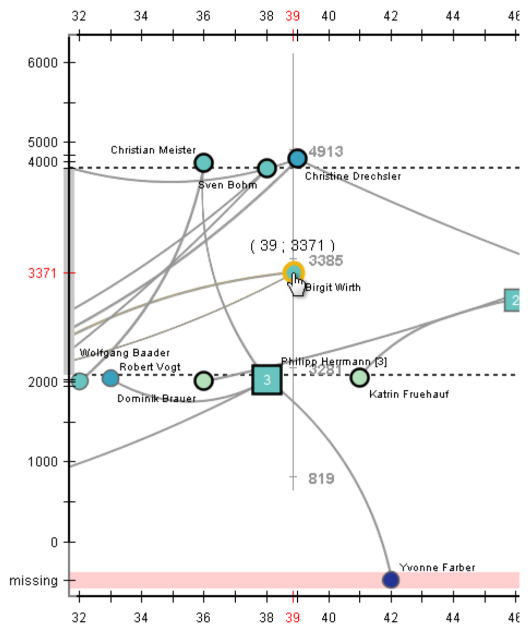
A suitable alternative particularly designed for qualitative data are semantic substrates [SA06]. The basic idea is to place nodes in distinct spatial regions that correspond to the values of one selected qualitative node attribute. Figure 5.6 shows a semantic substrates layout of the social network according to the node attribute OCCUPATION. Eleven values,



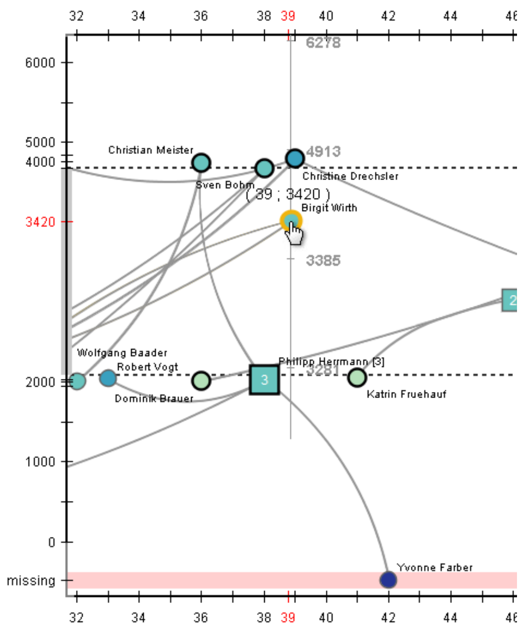
(a) Long press to start editing



(b) Drag coarsely toward target



(c) Activate focus+context lens



(d) Drag precisely toward target

Figure 5.4.: Editing a node's NET INCOME attribute value using a drag-and-drop gesture in combination with a local fish-eye magnification.

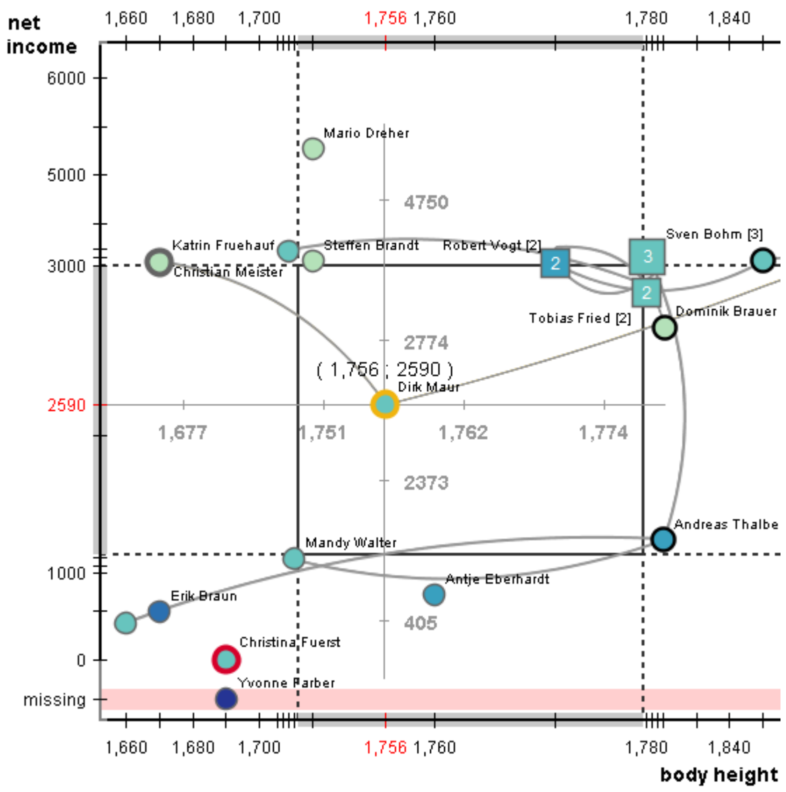


Figure 5.5.: Precise and simultaneous editing of two node attribute values (the NET INCOME and the BODY HEIGHT) of the person "Dirk Maur" by using the focus+context magnification on both axes.

including "technician" and "student", are encoded by individual regions. Bold labels are used for regions and smaller labels for nodes. Nodes whose attribute value is missing are collected in a dedicated twelfth region labeled *missing*.

While for a scatterplot layout, a node's quantitative value is encoded in the node's exact position, for a semantic substrates layout, a qualitative value is shown by the node's containment in a particular region. This difference has also consequences for editing interaction, as described later.

For semantic substrates, there is no fixed mapping of the regions to particular positions on the screen. Neither are the positions of nodes within the regions pre-determined. This allows optimizing the layout of regions as well as that of the nodes within the regions according to data characteristics, layout quality criteria, and application-dependent requirements. For example, for ordinal attributes, a region layout can be chosen that communicates order. For purely nominal data, for which no order exists, one can focus on graph aesthetics and employ a space-filling partitioning of regions [Rod+11], or a force-directed layout. In the examples presented here, a circular layout is used that reduces the number of edge crossings with regions. Within regions, one can again choose a

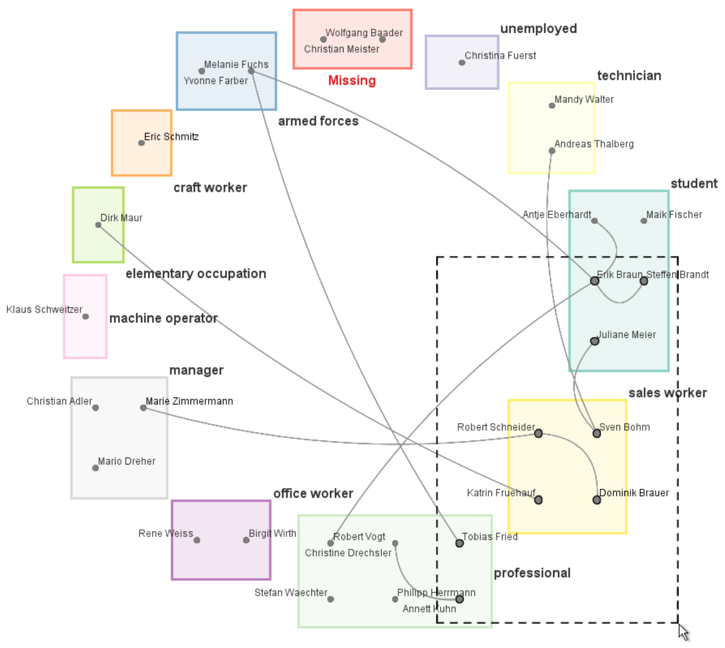


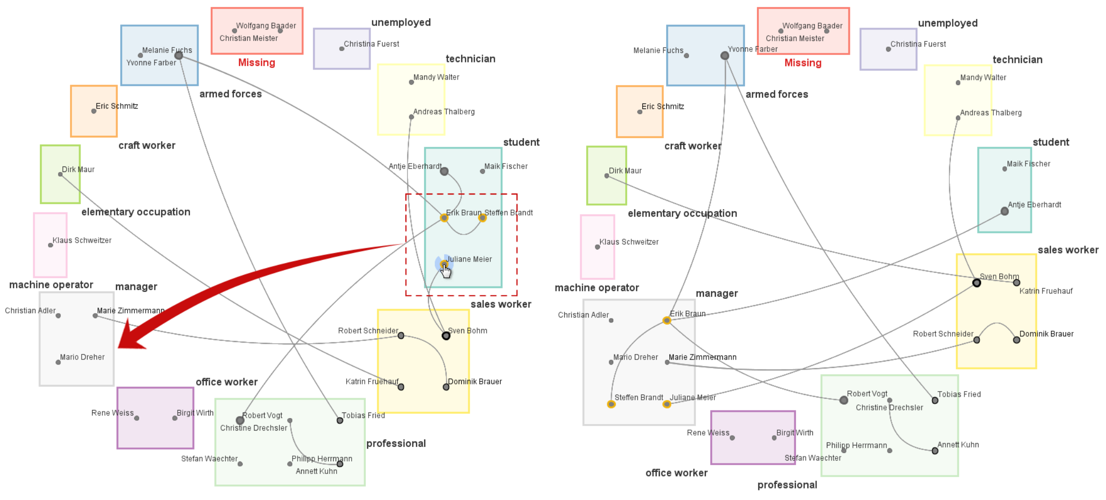
Figure 5.6.: Semantic substrates visualization according to the qualitative node attribute OCCUPATION. Its qualitative values correspond to regions, which are laid out on a circle. A node being displayed in a particular region indicates that the node exhibits the corresponding data value. Edges are filtered with respect to the currently selected nodes.

suitable layout, for example, a grid layout minimizing the average length of intra-region and inter-region edges as further illustrated in 5.6. Computing a force-directed node layout for every region would also be a possible alternative.

To estimate the value distribution without counting the number of nodes contained in every region, the size of each region additionally encodes the frequency of the associated value. Small regions with only few nodes could be interpreted as outliers being subject to editing.

Direct interaction and visual feedback Editing qualitative values can be reduced to literally dragging nodes from one region and dropping them in another; the exact drop position is irrelevant. Again, a long press is required to actually start the editing. Figure 5.7 (a) illustrates the interaction.

The result of editing a node's qualitative value is communicated visually by updating the affected regions (see Figure 5.7 (b)). Their size and internal node layout is re-computed to correspond to the new value frequencies. To avoid abrupt changes in the representation, the visual feedback is smoothly animated. To increase awareness of local and global outliers, the same highlighting strategy as for quantitative editing is used. For outlier detection, the strategy proposed in [LK01] is employed. Qualitative



(a) (b)

Figure 5.7.: Editing the qualitative OCCUPATION of three persons (indicated with a dashed, red rectangle) by dragging the dedicated nodes from the region "Student" to the region "Manager". (a) illustrates the interaction, whereas (b) shows the automatically adjusted representation after carrying out the editing.

outliers concerning a specific attribute are detected based on a fuzzy measure taking into account the total number of nodes having an attribute value, the total number of distinct attribute values and the frequency of individual values.

Reducing editing costs A cost factor when editing qualitative attribute values is to locate and reach the region of the target value. In contrast to quantitative attributes for which it is clear in which direction one will find the target value, the flexible layout of the semantic substrates does not provide such a naturally defined orientation. The costs are even higher, when not all regions are visible, for example due to previous zoom operations on the layout. In such cases, the user might not even see the target, nor be able to drop a node on it.

To keep interaction costs low without imposing any constraints on the graph layout, this approach couples the semantic substrates layout with an off-screen visualization. To this end, the semantic substrates view is extended by a border following the idea presented in [FD13] and [Jus+12]. The border is used to show proxies of those regions that are currently placed off-screen. A proxy serves three purposes. Its position and size provides orientation by indicating the direction and the distance of the corresponding region in the layout. The proxy is also used to display selected nodes based on node interestingness. How such an interestingness can be defined is discussed in Section 6.1 in detail. And finally, a proxy and its nodes can be interacted with and the result will be the same as if one would interact with the original region or node.

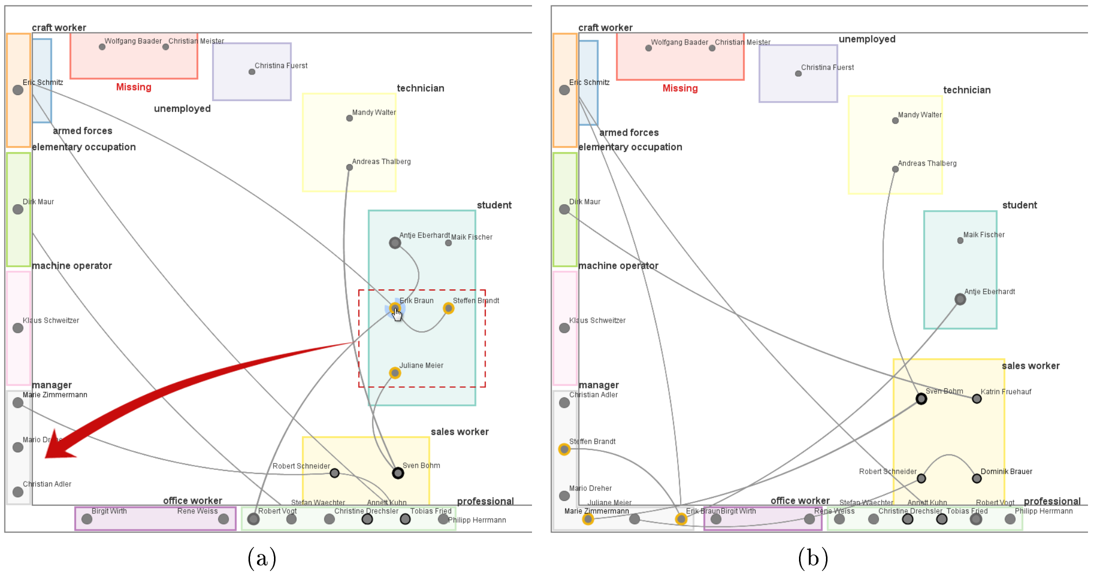


Figure 5.8.: Editing the OCCUPATION of three persons (indicated with a dashed, red rectangle) by dragging the dedicated nodes from the region "Student" to the proxy corresponding to the off-screen region "Manager". (a) illustrates the interaction, whereas (b) shows the automatically adjusted representation after carrying out the editing.

The proxies are computed according to a four-step heuristic algorithm:

1. Approximately determine the position and size of each proxy by projecting its off-screen region onto the border.
2. Detect overlaps among all projected proxies and resolve them by iteratively shifting and resizing the proxies.
3. Determine interesting nodes based on a degree-of-interest function.
4. Optimize proxy size and position so that important nodes can be displayed within the proxies while still maintaining the proxies' indication of direction and distance.

Once computed, proxies and interesting nodes are shown in the border around the view, as illustrated in Figure 5.8. Proxies help maintaining an overview of the data and they can also be used for editing. If a node must be moved to an off-screen region for editing its value, the node is dragged to the corresponding proxy. This interaction is illustrated in Figure 5.8 (a). Visual feedback is automatically provided by updating affected regions and proxies as depicted in Figure 5.8 (b). The interesting nodes displayed inside proxies can be edited in the same way as regular nodes. As all qualitative values (regions) are accessible at all times, additional navigation steps to reach off-screen regions are not necessary, which keeps the overall interaction costs low.

If a user wants to actually navigate to an off-screen region, its proxy can be used to trigger an automatic animation from the current view to the desired region, preventing manual navigation steps.

Taken together, the proposed techniques enable the user to edit quantitative and qualitative node attribute data. The next sections describe the implementation of the developed solution and illustrate how it can be applied in real-world scenarios.

5.1.3. Implementation

A proof-of-concept implementation of the proposed editing approach was realized by Christian Eichner and Arne Neuber within the presentation tool described in [Eic+15]. In this implementation, multivariate graph data containing several hundreds of nodes and edges can be loaded from csv formatted files.

The system's internal graph model as well as the graph visualization are based on the *JUNG* framework [Mad+05]. However, visualizing the data as described above required extending the functionalities of *JUNG* in several ways: The layout of nodes in both representations as well as regions in the semantic substrates layout required a new implementation. For computing node and region labels, a labeling algorithm was used [LSC08]. Moreover, an algorithm for dynamically computing overlapping nodes in the scatterplot layout, which must be represented as clusters (e.g., when nodes are dragged or the fish-eye transformation is used), had to be added. Here, a solution based on a simple disjoint-set algorithm was used. Last but not least, the interactive fish-eye transformation as well as off-screen visualization had to be implemented from scratch.

For displaying further visual elements like axes and their labeling, box plots and colored regions, the Java2D API was used.

All proposed interaction techniques for editing node attribute values are implemented and can be used via mouse+keyboard input.

5.1.4. Use Case and Evaluation

In order to test the proposed approach in realistic settings, the implementation was applied to two real-world use cases. For the purpose of evaluation, user feedback was collected within a user study.

5.1.4.1. Use Case

In this section, two application examples for visually editing quantitative and qualitative node attributes are presented.

Editing Quantitative Node Attribute Values of a Wireless Network The first example is about manual maintenance and correction of node attribute values of wireless networks.

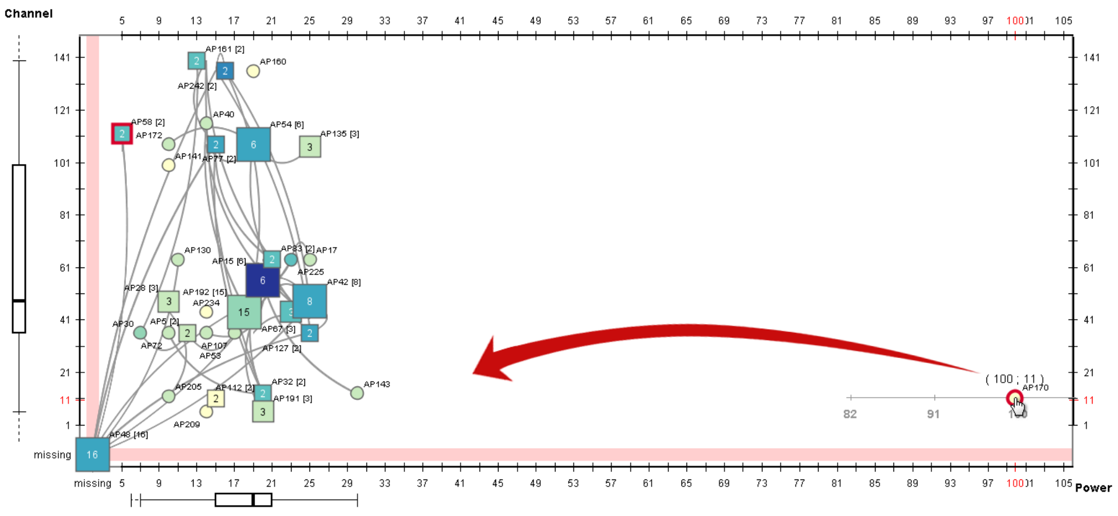


Figure 5.9.: Correcting the transmission POWER of an access point of a wireless network.

The data set contains information about a privately administered network (5 GHz frequency range) that has been established in a German city for private communication and file sharing. This network consists of 98 access points (nodes), which are identified by their subnet IP address, and 56 antenna connections between access points (edges). The access points are associated with two quantitative attributes: channel and transmission power in dBm.

Editing these attributes can become necessary for several reasons. For instance, if the channel of an access point overlaps with the channel of a newly installed wireless router located nearby, it makes sense to adjust the channel to avoid signal noise and low throughput. Another editing scenario arises when access points are removed from the network. To maintain network connectedness, it might be necessary to increase the transmission power of other access points. If such changes have to be applied, this can be done visually by using the proposed techniques.

As depicted in Figure 5.9, the network was visualized according to its two quantitative attributes as a scatterplot layout. The figure shows that the access points use different channels and that their transmission power is mostly set to 20 dBm. However, an extreme outlier with a transmission power of 100 can be recognized too. After checking with a maintainer of the network, he found out that this value was measured in mW instead of dBm and thus is erroneous. Using the novel editing approach, this value was corrected directly in the visualization by moving the access point to the position corresponding to 20 dBm (equals 100 mW). Additionally, access points with missing values are obvious at a glance. Once the owners of these access points report their channels, they can be inserted directly into the data as well. The benefit is, that the result of the data change is immediately visible. This enables the user to set edited values – for instance the transmission power – in relation to existing values for verifying them.

Editing Qualitative Node Attribute Values of a Document Network In a second example, the proposed approach was applied to dynamically created document networks. Such networks can be used to semantically organize documents (e.g., pictures, slides, and book pages) being part of a presentation. Documents correspond to the nodes of the network, whereas edges correspond to semantic relationships between documents. As a qualitative attribute, the documents are associated with a certain topic. This information is used to quickly identify groups of documents, which might be relevant for answering questions concerning a specific topic, or which might contribute to a discussion of a certain topic.

In the presentation tool described in [Eic+15], multiple users can add their documents to an existing document network in order to contribute to a presentation. These documents are then analyzed automatically to derive their topic (e.g., by parsing frequent keywords in a text or captions of a slide). However, there exist situations where this automatic derivation fails. Examples include slides with misleading captions or pictures without any text information. The results are documents with ill-classified or missing topic. In such situations, manual correction by the user becomes necessary. This usually requires to view the document first and to explore its semantic relationships.

For supporting both, the semantic substrates visualization according to document topics was used. To indicate document contents, nodes were represented as document thumbnails. Figure 5.10 shows an exemplary document network about EICHNER ET AL.'S presentation tool. As can be seen, documents with different document topics, for example "Infrastructure", "Use Case" or "Smart Room", exist. Documents with missing topic are placed in a dedicated region labeled "Missing". If details of a document must be viewed, the thumbnails can be used as shortcuts for opening the document. For correcting a document's topic (e.g., the missing topic of a picture), the proposed drag and drop editing interaction were applied to the document. If the correct topic did not exist, it was created in the graph layout beforehand.

With multiple users contributing documents (as envisioned in [Eic+15]), document networks can quickly grow to more than 100 nodes. In such situations, users have to zoom into detailed views in order to identify documents and their semantic relationships. This might cause some regions to be placed off-screen. However, with the proposed off-screen visualization, the editing of topics could be still carried out without expensive back-and-forth navigation. Moreover, as illustrated in Figure 5.10, the off-screen visualization provided support for identifying the semantically most related off-screen documents. This could be achieved by displaying off-screen documents with the strongest connection to documents in visible regions within the proxies. This is illustrated in Figure 5.10.

5.1.4.2. Evaluation

To evaluate this approach, user feedback sessions were conducted. The aim was to gather qualitative feedback concerning the usefulness and ease of use of the editing solution. According to the focus of this thesis, a focus was set on the novel editing techniques.

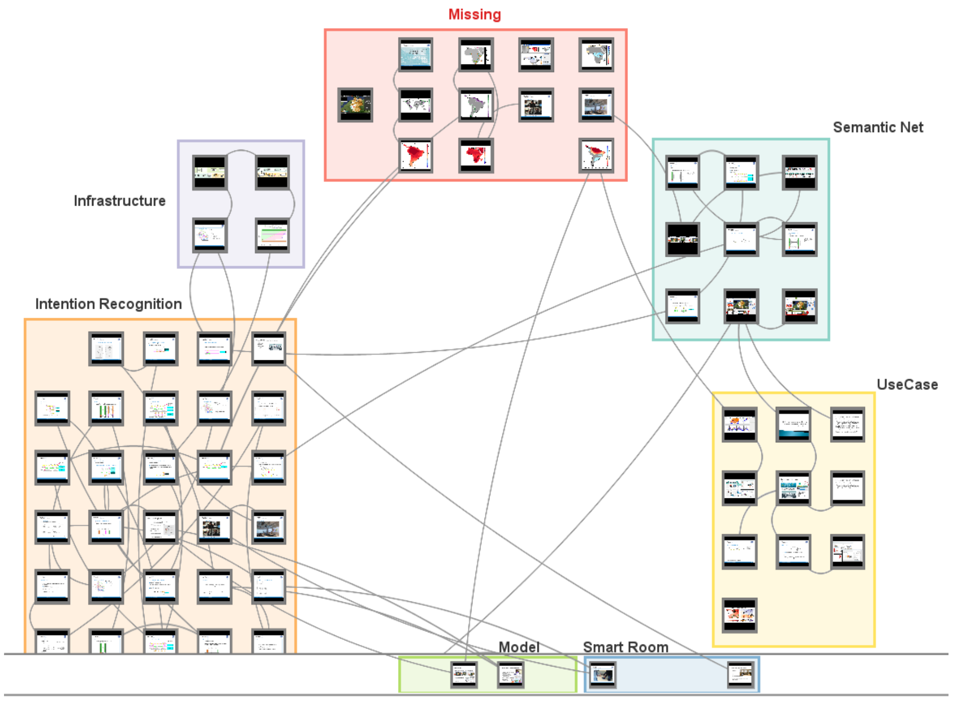


Figure 5.10.: A semantic substrates visualization according to document topics of a dynamically created document network about the presentation tool described in [Eic+15]. The proposed drag and drop editing techniques can be used to directly correct documents with missing topics placed in the region labeled "Missing".

Pilot Testing Before the actual sessions, comments were solicited from an expert in visualization and interaction. This pilot testing lead to minor adjustments of the implementation. These adjustments included improvements of the highlighting of selected nodes and the change of node sizes displayed in proxies for example.

Participants Ten persons were recruited (ages 24–58, two female). All of them were employees or students at the computer science department of a university. Six participants considered themselves experienced in interactive visualization. All participants were confident with mouse and keyboard interaction. None of them had used the editing techniques prior to the feedback sessions.

Procedure First the experimenter explained the data set - the small fictional social network (already shown in Section 5.1.2) with attributes, such as AGE, NET INCOME, and OCCUPATION. The dataset contained outliers and missing values.

The session continued with a demonstration of the editing in the scatterplot layout. The participants familiarized themselves with the visualization and the available interaction functionality. Afterwards, the participants were asked to carry out two editing tasks. The first task was to edit the AGE of a person to a particular value. For this task, editing precision was not a problem as the value range is narrow. In the second task, the missing NET INCOME of a person had to be set. The participants were prompted to chose a large value that is not an outlier value. For this task, they had to use the focus+context transformation in order to set the value precisely and they had to verify the outlier constraint from the visual feedback.

Thereafter, the experimenter switched the representation to a semantic substrates layout. Again the technique was demonstrated and the participants familiarized themselves with it. Two editing tasks had to be carried out. First, participants had to change the OCCUPATION of multiple persons from initially STUDENT to PROFESSIONAL, where the corresponding regions were visible on-screen. For the second task, participants had to edit nodes by moving them to a proxy at the view border.

During a session the experimenter took notes of any comments made by the participant. Individual feedback sessions took around 15 to 20 minutes.

Results All participants finished all task by using the new editing techniques. Generally, there was consensus about their usefulness. Valuable feedback was collected that confirms the applied design decisions and also makes suggestions for further improvements.

Several participants were surprised by the editing precision when using the focus+context technique. One of them commented "Using the local magnification is an intuitive solution to increase precision.”. Additionally, three participants acknowledged the benefit of constraining the drag to one axis for editing a single attribute value. Although outliers and missing values could be identified by all participants quickly, two participants suggested marking entire outlier regions in the background of the graph layout.

Further positive feedback concerning the semantic substrates editing could be received. One participant said that dragging nodes to different regions is easy and intuitive. Another commented that the proposed editing technique avoids setting incorrect data values, which sometimes happens when entering data values with the keyboard. The off-screen proxies could be used with ease. Yet one participant criticized that proxy size should encode value frequency, which would be consistent with regular on-screen regions. The animated visual feedback upon editing was received positively across all participants.

Some participants also made general suggestions for improvement. One participant said that standard undo/redo functionality is a necessity when editing data. Another valuable suggestion was to allow users to collect a number of data changes and to commit them as a single transaction. An interesting implication would be the ability to store annotated diff-files for data provenance.

5.1.5. Discussion

A novel approach for editing node attribute data directly in a visual graph representation was presented. Different attribute-dependent layouts were employed and dedicated interaction strategies for editing quantitative and qualitative attribute values were introduced. The attribute-centric visual representation supports the user in discovering values relevant for editing. Data modifications can be accomplished by directly interacting with the visual representation and their effects are communicated by visual feedback. This is a key advantage as cognitively expensive switches between the visualization and an external editing tool are unnecessary.

The proposed solution can be applied to diverse editing scenarios. It can help in manual data curation and the analysis of different *what-if* scenarios. First positive user feedback indicates that this editing approach is a promising alternative to standard raw-data editing techniques.

The proposed techniques addressed the editing by interacting with scatterplot and semantic substrates layouts. Yet, this editing approach is not restricted to these particular visualizations, but can be transferred to other visualization techniques as well. It is obvious that scatterplot matrices as well as many axes-based techniques (e.g., line charts or parallel coordinates plots) can be enhanced with similar direct editing mechanisms. Derived techniques might be even applicable for editing in tabular visualizations.

The presented solution cannot cope with extreme cases. Excessively large and small quantitative values, huge amounts of different qualitative values, or extreme value distributions (e.g., accumulation at singular points) will typically make visual editing difficult, if not impossible. Therefore, alpha-numeric input should always remain a fall-back editing solution.

A related question regards the duration of the editing. Right now, a definite answer cannot be provided due to many influencing factors: For entering a single value, alphanumeric input might be quicker. Yet, editing multiple values of a set of nodes might be performed faster when using the proposed approach. Moreover, when a target value is unknown and needs to be derived through experimentation, the novel approach can play out its strengths. The direct visual editing allows testing many different data values in a short period of time. This is analogous to the widely acknowledged advantages of direct manipulation for dynamic filtering.

The presented editing approach is limited to the node attributes which are currently encoded by the attribute-dependent layout. In the following section, an approach is presented which enables the user to edit attribute values of nodes *and* edges.

5.2. Integrating Radial Controls for Editing Graph Attribute Values

In the previous section, the visual editing of node attributes values based on graph representations with attribute-dependent layouts was investigated. However, for visualizing the graph's attribute values *and* the graph's structure, representations with attribute-independent layouts are also often used. Hence, it makes sense to consider such representations as a basis for editing attribute data visually as well.

In this section, a first attribute-editing approach on the basis of node-link representations with attribute-independent layouts is introduced. For visualizing node and edge attribute values potentially relevant for editing (e.g., missing values and outliers) a suitable mapping to the visual variables color and shape is used. For applying edit operations to qualitative or quantitative values, the user is free to integrate interactive editing controls into the representation on demand. When using these controls, visual feedback for the locally edited data value and changes to the global attribute value distribution is constantly provided. This allows verifying the effect of applied edit operations.

In the following, the specific problems and requirements for editing attribute values in attribute-independent layouts are described first. Afterwards, the novel approach is described in detail.

5.2.1. Problem Analysis

As described in the chapter introduction, the editing of attribute values is a three step procedure of (i) identifying data to be edited, (ii) altering the data and (iii) verifying the editing outcome. Now, solutions for each step with respect to the characteristics of graph representations with attribute-independent layout must be developed.

First, to determine attribute values relevant for editing (e.g., missing values or outliers), a suitable mapping from attributes to available visual variables must be found. However, in contrast to attribute-dependent layouts, attribute-independent layouts generally do not use a mapping to node or edge positions. Consequently, a mapping to visual variables like color, shape or area must be found. As described in [Tel07], this is a difficult task as multiple aspects must be considered:

Characteristics of the data Depending on the scale of the attribute (i.e., qualitative or quantitative), the mapping must use a compatible visual variable which can represent the data effectively [Mun14]. Moreover, the mapping should consider statistical features like the value distribution including extreme values.

Characteristics of the tasks As presented in [TFS08b], a mapping must be selected which facilitates the user's task at hand. To communicate values potentially relevant for editing (e.g., outliers and missing values), the mapping should effectively support their localization. This is difficult especially for missing values, as there are only a few approaches in literature. To support the user in recognizing data changes caused by edit operations, the applied mapping must additionally enable the user in identifying individual values.

Characteristics of the user As the visual perception of users can vary (e.g., some users might suffer from color blindness), a mapping should be selected which goes hand in hand with the capabilities of the user.

Characteristics of the output device This aspect is mostly relevant for color mappings, as different output devices use different systems for defining and displaying color [TFS08b]. However, varying display resolution also plays a role when mappings to the visual variables size or area are applied.

For selecting appropriate color mappings with respect to these aspects, several approaches already exist [TFS08b; STS05; HB03; BRT95]. However, for selecting mappings to alternative visual variables, only few literature is available to build on. The selection of multiple visual variables for communicating multiple attributes at the same time is an additional challenge as visual variables might interfere pair-wise [Mun14].

Second, interactive means for carrying out the editing of attribute values must be provided. In this regard, the use of representations with attribute-independent layout poses additional challenges. In contrast to representations with attribute-dependent layout, the editing of attribute values cannot be reduced to directly moving nodes or edges in the graph layout. Instead, novel editing interactions based on direct manipulation [Shn83] must be developed. This however is not straight-forward as direct manipulation is usually implemented as a spatial modification of a graphical object.

Third, in accordance with the base representation and the interaction techniques used to edit the data, appropriate visual feedback for analyzing and verifying the editing outcome must be integrated. A requirement is the communication of changes to individual data values as well as the global value distribution.

Taking these requirements into account, a novel approach to visually edit a graph's node and edge attribute values was developed. In this approach, *radial menus* play an important role.

Radial Menus Radial menus are context menus that display menu options in a circular layout around an object of interest. The selection of options depends on direction and radius instead of distance compared to standard linear context menus. This is the reason why radial menus can outperform linear menus in some settings [Cal+88].

In the context of visualization, radial menus have been used to select and steer parameters of data representations (e.g., scatterplots of multivariate graphs [Via+10]), to select subgraphs in node-link representations and to modify their layout [MJ09]. In context of human-computer interaction, radial menus have been studied extensively and different types have been developed [Tha+14; BA11; Mcg+02; GW00; Poo+00; Kur+99].

Even in video games for desktop computers (e.g., *Mass Effect 3*) or smartphone apps (e.g., *Twitter* app *Twheel*) radial menus can be found. However, they have not been considered in the context of visual graph editing before.

5.2.2. Approach

The key idea of this approach is to enable users to edit an attribute value of a node or edge through interaction with a pop-up interface related to a context menu. This interface provides generic controls for editing both, qualitative and quantitative attribute values and is equipped with visual cues providing feedback upon editing. In contrast to editing node attribute values by moving nodes in the graph layout, the editing here is accomplished *in place* without changing the graph layout.

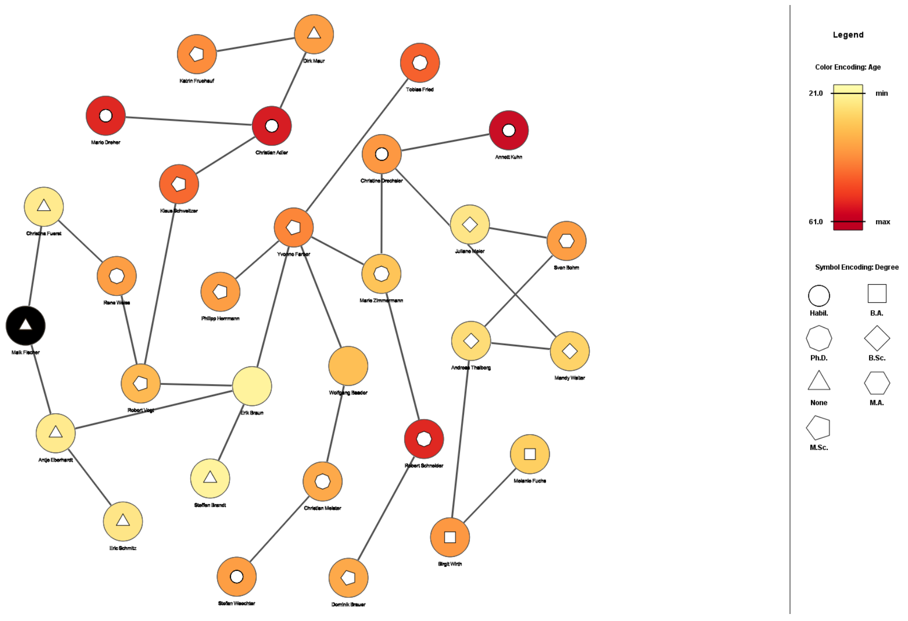


Figure 5.11.: Node-link representation with force-directed layout of the fictional social network from Section 5.1. The persons' AGE is currently encoded by varying node color. Black color indicates a missing value. The persons' highest academic DEGREE is depicted by varying symbol attached to nodes. Persons with missing DEGREE are shown as nodes without any symbol. Edges represent friendships between persons.

The three components of this editing solution, i.e., (i) the base visualization, (ii) the direct editing interaction and (iii) the visual feedback, will be described next.

5.2.2.1. Base Visualization

Node-link representations with attribute-independent layout form the basis for visualizing the graph's structure and the graph's attribute values. The structural aspect is communicated with the graph layout, whereas nodes are depicted as circles and edges are represented as lines. To minimize layout quality issues like node overlaps or edge crossings, a suitable graph layout algorithm is applied. Depending on the application domain this algorithm might differ. Figure 5.11 shows a node-link representations with a force-directed node-layout of the fictional social network from Section 5.1.

Attribute data associated with nodes or edges is encoded by varying appearance of nodes and edges. For communicating node attributes, the visual variables color and shape are used. The visual variable color is compatible with both, qualitative and quantitative attributes. Shape is only effective for communicating nominal data [Mun14]. Thus, one qualitative and one nominal node attribute or one quantitative and one nominal node attribute can be visualized at the same time. A redundant mapping of one nominal attribute is possible as well.

If a qualitative node attribute has to be communicated with color, a mapping to a suitable qualitative color scale is applied [HB03]. Thereby, individual attribute values are mapped to individual colors. Moreover, the color scale should fit the capabilities of the user and characteristics of the output device. If a quantitative node attribute is visualized with the nodes' color, a mapping to a quantitative color scale is applied [BRT95]. Such mappings enable users to identify different attribute values and to recognize value changes caused by edit operations.

To support the localization of outliers in quantitative attribute scales, a linear mapping is used so that a certain distance between data values is reflected by a proportional distance between assigned colors. This way, the color of outliers can be distinguished clearly from the color of regular values.

To enable the identification of missing values, respective nodes are colored with sharp contrast to the colors of the color scale used to visualize existing values. If for instance a yellow to red color scale is used and the background color of the node-link representation is white, coloring nodes with missing values in black is one suitable option. In Figure 5.11, the AGE of persons is visualized in such a way.

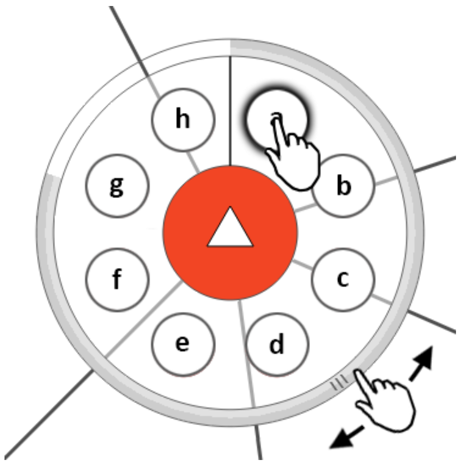
For communicating a nominal node attribute with the visual variable shape, additional symbols are attached to the center of nodes. For different nominal values, symbols with different shape are used. As depicted in Figure 5.11, such shapes can be regular polygons with different amount of corners. For nodes with missing values, no symbol is attached. This clearly communicates the absence of a node's attribute value.

For the communication of edge attributes, a similar mapping strategy is pursued. A mapping to color is used to communicate a qualitative or quantitative edge attribute. Small symbols (e.g., connector ends like in UML class diagrams) are additionally attached to edges for communicating a second nominal edge attribute. However, to be able to distinguish between node and edge attribute values, the used color scales and symbols should differ.

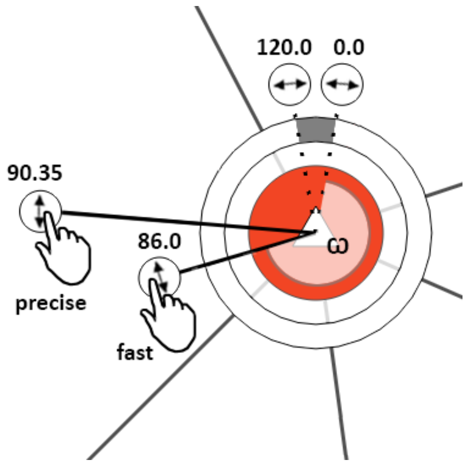
The visualization of a third node and edge attribute would be possible by additionally varying a node's and edge's area. The visual variable area however interferes with color if nodes become too small or edges become too thin [Mun14]. For that reason, such a mapping should only be applied if it really necessary to visualize three node or edge attributes simultaneously.

To enable the user to decode attribute values according to a node's or edge's appearance, a legend is always displayed.

The proposed base visualization is one possible option. Alternative representations like node-link representations using glyphs or matrix representations with varying cell appearance might be applied as well. The identification of individual advantages and disadvantages is beyond the scope of this work, since the editing interaction proposed next is applicable to different mappings.



(a) Radial menu



(b) Radial slider

Figure 5.12.: Basic features of (a) radial menus and (b) radial sliders. Radial menus allow selecting individual options of a finite set of options, whereas radial sliders enable users to select individual values of a quantitative value range.

5.2.2.2. Direct Interaction

If a certain value to be edited is determined, this approach enables users to invoke an editing interface related to a context menu on the node or edge of interest. The editing interface is automatically integrated into the representation and especially designed for touch input. Similar to a standard context menu, this interface consists of different levels, which supports the user's tasks step by step. These levels are either *radial menus* or *radial sliders*:

Radial Menu Radial menus are used to support tasks where specific options must be selected. They present selectable options as menu entries in a circular layout around the node or edge of interest. In order to reduce occlusion with graph elements located nearby, menu entries are presented with small circles instead of pie segments. The size of these circles is set according to existing conventions for touch interaction (10mm diameter) [Wro10]. As illustrated in Figure 5.12(a), selecting an option can be done by tapping the associated menu entry. Once selected, menu entries are highlighted with a halo.

Since radial menus display only eight menu entries effectively, a radial scroll bar is added in cases where more entries need to be presented. Similar to common straight scroll bars, users can drag either the scroll bar or the menu entries in a radial fashion to view further entries (see Figure 5.12(a)).

Radial Slider Radial sliders are used to support tasks where values of a quantitative value range must be selected. As depicted in Figure 5.12(b), these controls are presented as a ring around the node or edge of interest. The slider handle can be dragged in a radial fashion (indicated by small arrows) to set different values of a default value interval including minimum and maximum. Which value is set depends on the angle ω, which is defined by the spatial position of the slider handle. In case of a discrete value range, (e.g., integer values), certain angular intervals correspond to individual values. In case of a continuous value range, individual angles correspond to individual values. An angular interval depicted in gray is associated with missing values. The currently selected value is visualized with a label attached to the outside of the slider handle (not occluded by the finger tip). Similar to a standard slider, the first and the last value of its value range are additionally depicted as labels. Additional ticks and labels to communicate further values (e.g., like an axis labeling on a scatterplot) might be helpful.

As discussed in Section 5.1.2.1, a key concern when inputting quantitative values via spatial modification is interaction precision. When moving the slider handle by one pixel, the angle ω changes by a constant delta and thus also the corresponding quantitative value. However, this delta might be too large for selecting fine-grained values. To allow for sufficiently small deltas, one can provide a zoom mechanism for varying the value interval of the slider. This, however, would require to zoom in and to zoom out multiple times when different values must be selected.

The slider's mechanism to define values allows for an alternative solution to increase precision. The covered distance of a slider handle drag can be described as the arc length L of a circle segment, which is defined by the length of the slider handle (i.e., the circle radius r) and the change of ω, i.e., $\Delta\omega$: $L = 2 \cdot r \cdot \pi \cdot \Delta\omega / 360$. A key observation is, when r is multiplied by a certain factor t, L is also increased by t. Thus, the drag distance to achieve a certain $\Delta\omega$ increases with increasing slider handle length. This in turn means, with longer slider handles, more fine grained values can be selected [PD15; Mül+14]. Consequently, users are enabled to pull out the slider handle during drag. Depending on the value interval mapped to the slider, the value precision (i.e., the affected decimal place) is adjusted gradually if r exceeds certain thresholds. This way, short handles allow to set values fast (but less precise) whereas long handles allow to set values more precise (but slower, due to the increased drag distance). This is depicted in Figure 5.12(b). If the gained precision by pulling out the slider handle is not enough, an additional local magnification similar to 5.1.2.1 can be used.

If a value outside the default value interval must be set, the slider's value interval can be increased. This can be done by dragging slider handles representing the lower and upper bounds of the default value interval (marked with dashed lines in Figure 5.12(b)). If for instance the slider handle of the upper bound is dragged counter-clockwise, the angular interval for the default value interval is compressed. The gained space is then automatically mapped to additional values above the default upper bound. Space for additional values below the default lower bound can be made available analogously.

Similar to the visual design of a radial menu, a radial slider requires only little screen space and thus keeps occlusion of nodes and edges located nearby low.

The editing of a node or edge attribute value requires the user to carry out the following tasks one after another:

1. Select node or edge of interest.
2. Select attribute value to be edited.
3. Define new target value.

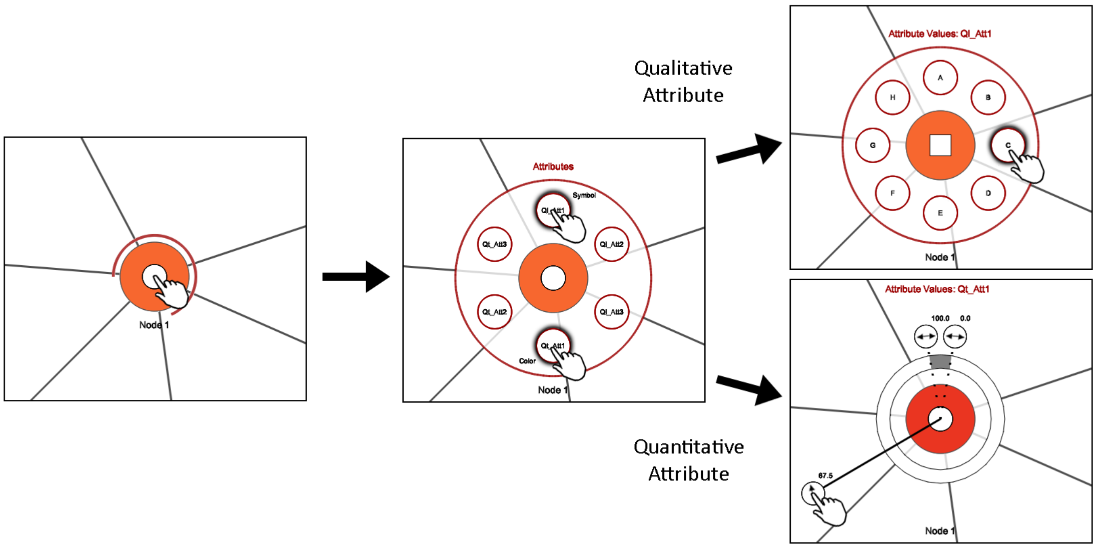
The described radial menus and radial sliders are now combined to an editing interface, which allows the user to accomplish these tasks one after another: Since the whole approach can be applied analogously to edit node or edge attribute values, it is described for node attributes exemplarily.

Select Node of Interest If an attribute value of a specific node must be edited, the editing interface must be invoked on this node. For doing so, a long press must be applied to the node's representation for a certain time (e.g., 0,5s). This is illustrated in Figure 5.13 (a). The reasons for choosing a long press are the same as described in Section 5.1.2.1. To communicate the progress of the gesture, visual feedback is provided through a progress bar around the user's fingertip. As soon as the long press time has elapsed, the first level of the editing interface is presented automatically around the pressed node. In order to set the user's focus to the pressed node, the view is automatically centered on the node's position. This step is illustrated in Figure 5.13 (a).

Select Attribute Value to be Edited The first level of the editing interface is a radial menu presenting available attribute values as menu entries. Menu entries for node attribute values which are currently represented by the node's color, symbol (or area) are highlighted with an additional label. This visual cue can help in situations where users identified an attribute value to be edited based on the nodes visual appearance, but do not remember the current encoding. To select an attribute value which need to be edited (i.e., to carry out a *Select or Specify Data to Edit* task), the user simply taps on the dedicated menu entry. This interaction is represented in Figure 5.13 (b). Afterwards, the second level of the editing interface appears automatically.

Define Target Value If a value of a qualitative attribute has been selected, a new radial menu appears. As shown in Figure 5.13 (c) top, this radial menu presents values of the selected attribute as individual menu entries. The currently set value for the node of interest is highlighted if existent. With a simple tap on a different menu entry, user's can set a different attribute value for this node (i.e., accomplish an *Apply Edit Operation* task). Doing so might require scrolling if more than eight different attribute values exist.

If a value of a quantitative attribute has been selected, a new radial slider appears. As illustrated in Figure 5.13 (c) bottom, this radial slider shows the current attribute value of the node of interest. To edit the node's selected attribute value, users have to drag the slider handle. If attribute values outside the slider's value range must be set, the user must increase the slider's value range first as described above.



(a) Select Node (b) Select Attribute Value (c) Define New Value

Figure 5.13.: Step-by-step editing of attribute values using the editing interface. (a) invoking the interface on the element of interest. (b) selecting the attribute value to be edited. (c) editing qualitative values with radial menus and quantitative values with radial sliders.

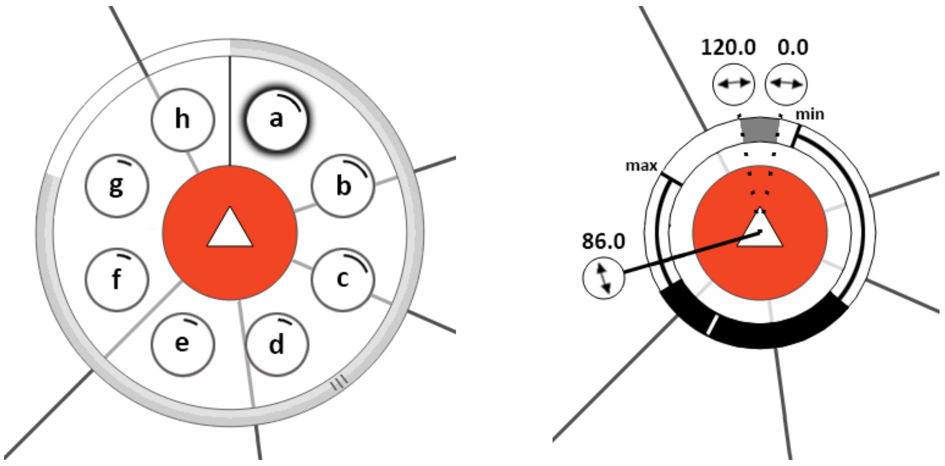
5.2.2.3. Visual Feedback

Providing visual feedback upon editing is important to enable the user to comprehend the changes to the data. For that reason, the nodes visual appearance is immediately refreshed if new attribute values have been selected with a radial menu or slider. This way, the changes to the locally edited data value become obvious.

To further enable users to analyze the effect of update operations on the global value distribution, radial menus and radial sliders for editing are additionally equipped with visual cues. These visual cues are constantly refreshed during editing.

For radial menus, two different options were developed. The first option is to equip menu entries with an additional ring segment which encodes the global frequency of the dedicated value with its length (see Figure 5.14(a)). These visual cues can be used to compare the value distribution before and after updating a value in order to obtain the global editing effect. Moreover, the user is able to set the desired target value in relation to attribute values of further existing nodes.

The second option is to equip menu entries with a filling level similar to a bar chart for communicating the frequency of the dedicated value. Similar to the ring segments, these visual cues provide feedback to analyze the editing effect on the value distribution. Which option is better suited has yet to be determined.



(a) Radial menu (b) Radial slider

Figure 5.14.: Integrated visual cues to communicate the editing effect: (a) radial menu entries with ring segments visualizing the frequency of individual qualitative attribute values and (b) radial sliders with a box plot and markers showing the value distribution as well as the current minimum and maximum values of a quantitative attribute.

As illustrated in Figure 5.14(b), radial sliders are equipped with a box plot which informs the user about the current attribute value distribution. For enabling the user to set newly defined values in relation to existing values, the whiskers of the box plot are used to communicate the current minimum and maximum values. In contrast to representations with scatterplot layout, these information might not be visible at a first glance by just looking at the base representation. If quantitative values are edited, the box plot visualizes changes of the value distribution immediately. Moreover, produced outliers are indicated.

The editing interface offers additional interactive means to navigate to previous interface levels without costly re-invocation. These and the visual feedback during menu navigation are described next.

Interface Navigation

Every level of the editing interface provides consistent ways to navigate to the previous interface level. This can be necessary for example when multiple attribute values of a node must be edited or when a node attribute value has been selected by mistake. The gesture to be applied is a single tap on the node of interest. To close the editing interface at once, a tap into free space has to be applied. This will also restore the view from where the editing interface has been invoked.

To make the user aware of appearing, disappearing or changing interface elements, animations are used. For example, when radial menus or radial sliders are opened, they appear with a grow-animation from the backside of the node of interest. When they are closed, they reversely disappear with a shrink-animation.

The visual feedback, the focus on the objects of interest and the visual cues to provide awareness of the editing effect follow the concept of fluid interaction [Elm+11].

Supporting *Adjust Representation* tasks

The new editing interface enables users to edit arbitrary qualitative and quantitative attribute values independently of the base representation and provides visual feedback for verifying the effect of applied data changes. However, when the values that have to be edited must be determined via exploration, the base representation has to communicate the attributes currently of interest. As the number of attributes encoded by the base representation is limited, the need for adjusting it might arise. Usually, external GUI controls are provided for this purpose. This however requires users to leave their working context (the node-link representation) which interrupts the workflow. Similar to the editing of graph attributes, the idea is to adjust the encoding through interaction with the visual representation. For this purpose, an on-demand interface consisting of radial menus can be used again.

Let $A = \{A_1, .., A_p\}$ be the set of existing node attributes and let $W = \{W_1, .., W_q\}$ be the set of existing edge attributes. To adjust the encoding of a visual variable, the user has to perform the following three steps:

1. Select the encoding of nodes or the encoding of edges
2. Select a visual variable *Var*
3. Select a node attribute $A_i \in A, 1 \leq i \leq p$ or edge attribute $W_j \in W, 1 \leq j \leq q$ compatible to *Var* so that a default mapping $m : A_i \rightarrow \mathit{Var}$ or $m : W_i \rightarrow \mathit{Var}$ can be established

Because all three steps require selecting an option from a finite set of options, the accomplishment of these tasks is supported by an encoding interface consisting of three levels of radial menus. For invoking this encoding interface, the user has to apply a single tap into white space. The selection of menu entries and the menu navigation is identical to the editing interface. However, to communicate the difference between encoding and editing interface, the interfaces are colored differently. For the encoding menu, green color is used. The editing interface is colored in a signaling red to make the user aware of crucial data editing actions.

5.2.3. Implementation

For testing and gaining user feedback, the proposed approach for editing nodes and for configuring the encoding was implemented in a prototypical java application for a desktop multi-touch screen. Radial menus and radial sliders can be used as described for editing attribute values. As a basis for implementing touch gestures and representing the data, the *MT4J* platform [LRW10] was used. Besides touch interaction, standard mouse interaction can be used alternatively.

The node link representation is embedded into a zoomable space where pan and zoom navigation can be used with standard drag and pinch gestures. To help novice users in interacting with the editing (and encoding) interface, the system automatically provides textual interaction hints depending on the current situation.

All proposed animations for interface interactions are all implemented. By using $smoothStep(x) = 3x^2 - 2x^3$ as interpolation function for computing animation steps, animated object translations, rotations, enlargements or reductions become smooth.

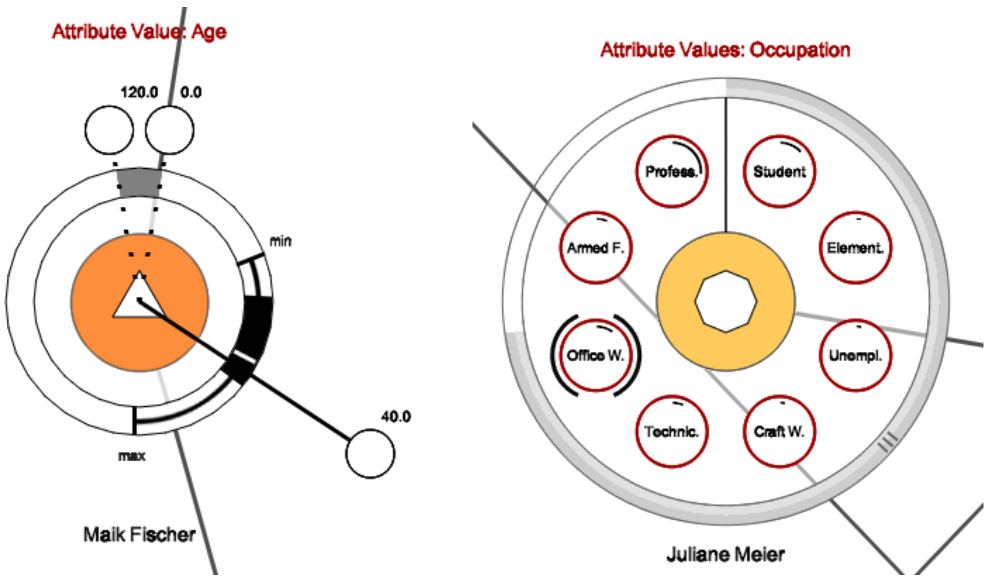
5.2.4. Use Case and Evaluation

The proposed approach was applied to edit personal data of a social network and user feedback was collected for evaluation purposes.

5.2.4.1. Use Case: Editing Node Attribute Values of a Social Network

A use case for the proposed approach is the manual editing of personal data in a social network. In such networks, nodes usually represent persons and edges depict some kind of relation between persons, for instance friendship. Often, there exist additional information associated with persons of these networks. An example is the user profile data of *Facebook*. That information can be interpreted as node attribute values. Whereas some of them are set automatically and remain invariant (e.g., the date of creating the user profile), some attribute values can be changed. In this regard, attribute values can be distinguished which need to be edited from time to time (e.g., a person's occupation) and attribute values which need to be set once in the course of completing the user profile (e.g., the persons age, nationality and gender). As both situations require editing individual attribute values of nodes, the proposed approach can be utilized.

To demonstrate the applicability of the novel editing and encoding interfaces, node attribute values of the fictional social network from Section 5.1 have been edited. This network consists of 30 persons, which are represented as nodes and 30 friendships, which are depicted as edges. Several qualitative and quantitative attributes are associated with the persons including AGE, BODY HEIGHT, HIGHEST ACADEMIC DEGREE and OCCUPATION. Although this network is small, it is suitable to demonstrate the editing techniques. Using the prototypical implementation, the network was visualized as a node-link representation where node attributes are encoded with corresponding colors and symbols at the nodes. Initially, the persons' AGE was encoded by color and the HIGHEST ACADEMIC DEGREE was encoded by symbol. The resulting representation is shown in Figure 5.11 on page 100. The black representation of the node "Maik Fischer" revealed that this person's age was missing. Adding this value could be carried out easily by invoking the editing interface on this node, selecting the attribute value AGE in the radial



(a) Editing a quantitative node attribute value (b) Editing a qualitative node attribute value

Figure 5.15.: Editing node attribute values with the proposed editing interface. (a) using a radial slider to add the missing AGE of a person (quantitative node attribute value). (b) using a radial menu to update the OCCUPATION of a person (qualitative node attribute value) from "Student" to "Office Worker". Note that the selection highlight of radial menu entries still needs to be replaced with the described halo to avoid visual interference with the ring segments for communicating value frequency.

menu, and eventually setting the value by interacting with the radial slider as depicted in 5.15(a). Afterwards, the persons' OCCUPATION was analyzed. For that, the encoding of the representation was adjusted. For doing so, the encoding interface was invoked and OCCUPATION was mapped to symbol. Afterwards, the occupation of the node "Juliane Meier" was updated from "Student" to "Office Worker". This required to invoke the editing interface on this node, to select the attribute value OCCUPATION, and to define the new target value "Office Worker" in the radial menu. The last step of this procedure is illustrated in Figure 5.15(b).

5.2.4.2. Evaluation

With the help of the prototypical implementation and the described social network, a user study was conducted to evaluate the editing approach. The first objective was to assess whether users are able to use the editing and the encoding interface in order to accomplish editing tasks. The second objective was to gather feedback concerning the suitability of the interface representation and interface interaction.

Pilot testing Before the actual study began, two pilot testers checked the study's feasibility, the appropriateness of session times needed and whether questions are suitable for the objectives. The initial feedback lead to minor adjustments of the order in which questions were asked.

Participants 14 persons (two female, age 23 and 28, twelve male, ages 23-35) participated in the user study. All were employees or students at a university and judged themselves as experienced in visualization. All of them were familiar with touch interaction. None of the participants had used the prototype prior to the study. Attendance on the study was voluntary and the participants received no compensation.

Tasks and devices Six tasks were selected which required users to work with radial sliders and radial menus of both, the editing interface and the encoding interface. Included tasks where to explore the social network from the use case by using pan and zoom navigation, to change the color and symbol encoding of nodes and to edit qualitative and quantitative node attribute values.

The study was performed using a 23 inch, capacitive multi-touch screen in a slightly tilted, horizontal position. A reference sheet with the interaction techniques was placed next to the screen so that participants could recall specific touch gestures when needed.

Procedure Every study session started with obtaining background information including age and touch interaction experience. Then participants were briefly introduced to the topic, the test data set was explained and the software was demonstrated.

Thereafter, the actual study started. The users could decide on their own, whether they wanted to sit or stand in front of the touch screen. The first part was a five minutes training time, where the participants were asked to try out the interaction techniques on their own. In the following, the participants had to solve the six tasks. Each task was given as an oral instruction. While the participants were working, they were instructed to describe what they were currently doing. Moreover they were asked about their impression concerning the interaction techniques. At all times, the participants could ask for assistance. After finishing all tasks, a prepared questionnaire was used to gather further information. The experimenter made sure that all questions had been answered prior closing the session. Each session took about 35-40 minutes including 15 minutes for the introductory part and the training time.

Results All participants solved all tasks. The overall feedback concerning the editing approach was positive and several participants acknowledged its usefulness. One participant additionally mentioned that it could be easily extended to support further tasks, for instance applying edit operations to sets of nodes or edges. There were only few critical comments how the approach could be improved or extended. In the following, the feedback concerning the interface representation and interface interaction is summarized.

Interface representation All participants confirmed that the representation of both interfaces is suitable. One comment was: "By embedding editing controls around a graph element, it becomes clear which element is affected."

According to the question whether the interfaces levels are logically ordered, only positive feedback was received. One participant said that "the interface levels were ordered as expected". To better facilitate menu navigation, two participants proposed to add visual cues to every level describing what can be done in the previous and the subsequent level. A further valuable suggestion was to order entries of radial menus lexicographically to find individual ones faster.

The users were further asked whether they could distinguish between the encoding and the editing interface. Eleven participants answered this question with yes, whereas three participants were undecided. These three mentioned there were situations at the beginning where they were uncertain about the purpose of each interface. This indicates that the visual feedback in this regard must be improved.

The last question concerning the interface representation addressed the animations. The majority of participants considered them as helpful. "These smooth animations clarify what is currently happening. Abrupt changes would not be comprehensive".

Interface interaction All but one participants mentioned that they could easily work with both interfaces using touch-interaction. However, three participants could imagine that the editing interface becomes ineffective when a large number of attribute values must be edited. In such cases they would prefer the traditional editing procedure in tabular representations.

Positive feedback was given concerning the varying precision for value selections with radial sliders. Two users highlighted the benefit of changing the precision during a single drag: "It is great that you can skip value intervals fast with short slider handles. When you roughly reached the value you want to select, you can simply pull out the handle for precise value selection." Although not expected, one user even said: "Right from the start it was clear that the handles distance can be used to adjust precision". Moreover, one user positively commented that the automatic view centering upon interface invocation provides maximal space for pulling out the slider handle.

5.2.5. Discussion

In this Section, a second approach for visually editing attribute data of nodes and edges was presented. This time, node-link representations with attribute-independent layout were used to support the user in discovering values that are relevant for editing. The editing of qualitative and quantitative attribute values is accomplished in place by interacting with an editing interface invoked on a node or edge of interest. The effects of local and global data changes upon editing are communicated with visual feedback. Furthermore, it has been described how a similar pop-up interface can be used to adjust the encoding of the visual representation on demand. This further relieves the user from using external controls.

To overcome the indirection of classic mouse+keyboard interaction, touch interaction was utilized. The used touch gestures are highly compatible with an existing gesture set for exploring and editing the graph's structure [FHD10], as well as the gestures used for the *EditLens* approach from Section 4.1. Thus, these approaches can be used side by side. Although the interfaces can be interacted with quite fast, they still can be improved in this regard. For that, the variant of radial menus called *marking menus* [LGF10; KB94] could be considered. It might even be possible to replace radial sliders with adapted marking menus so that the editing of an attribute value can be carried out with a single drag gesture. The verification of this hypothesis is left for future work.

A prototypical implementation was used to edit personal data in a social network and thus demonstrates the applicability of the proposed approach. Its usefulness was positively evaluated within a user study.

Although this approach was introduced using the example of node-link representations with attribute-independent layout, it can be applied similarly to alternative representations with attribute-independent layout (i.e., matrices or implicit representations). A precondition is that the elements to be edited are explicitly represented so that the editing interface can be invoked on them. The proposed approach is not suitable for editing node attribute values represented by an attribute-dependent layout, as the editing could lead to abrupt and confusing changes of the graph layout.

A limitation of this approach is that it addresses the editing of single attribute values of single nodes and edges only. However, as proposed by a participant of the user study, it might be possible to extend the approach to cover the editing of attribute values from a set of selected nodes or edges. Further limitations are associated with the use of radial menus and radial sliders. While radial menus can effectively display up to eight options, they become ineffective (even with additional scrolling) when a large number of options must be displayed. Such situations occur if the graph has a large number of attributes or if qualitative attributes have a large value range. There are also situations where radial sliders reach their limits. Examples are excessively large continuous value ranges were even the proposed techniques to increase precision become insufficient. For that reason, it is important to provide alpha-numeric input as fall-back solution for editing.

5.3. Summary

In this chapter two novel approaches supporting the visual editing of node and edge attribute values were introduced. Both approaches support the user in the carrying out the three necessary steps when editing attribute data: (i) identifying data to be edited, (ii) altering the data, and (iii) verifying the editing outcome. The reference of these steps to the tasks of visual graph editing described in Chapter 3 was established.

The first approach introduced in Section 5.1 addresses the editing of node attribute data in node-link representations with attribute-dependent layout. Scatterplot layouts are used as a basis for editing quantitative attribute values whereas semantic substrates layouts are employed if qualitative attribute data must be changed. Depending on the layout, dedicated editing interaction based on drag and drop of nodes were developed. Visual feedback for communicating local changes to individual values as well as global changes to the value distribution were proposed.

The second approach presented in Section 5.2 addresses the visual editing of node and edge attribute values in representations with attribute-independent layouts. Using the example of node-link representations, a pop-up interface for editing qualitative or quantitative attribute values was developed. It can be invoked on a node or edge of interest on demand and allows the editing in place without changes to the graph layout. Visual feedback for enabling the user to verify the editing outcome is provided similarly to the first editing approach.

The applicability of both approaches was demonstrated with a use case. Moreover, positive user feedback for both approaches was collected within user studies.

As the proposed editing approaches are based on different graph representations and address certain attributes only, switches between these representations might become necessary for flexibly editing all aspects of the graph's associated attribute data. To preserve the user's mental map, it is important to provide a seamless and comprehensive transition when switching between different representations. A suggested solution in this regard is to use animations.

The Chapters 4 and 5 introduced novel techniques for visually editing the graph's structure and the graph's attribute values. Thereby, support for the user's tasks was focused. However, there might be situations where not all aspects of the data are visible at the same time. Besides providing means for adjusting the encoding of the representation globally, there exist further options for supporting the user in this respect. Such options are investigated in Chapter 6.

6. User Support for Visual Graph Editing

The previous two chapters introduced new techniques for editing a graph's structure and a graph's attribute values. However, as these techniques are based on visual graph representations, limitations are possible with respect to two aspects: the available screen space and the size and complexity of the graph. On the one hand, visualizing larger graphs often lead to representations which do not entirely fit into the available screen space at a sufficient resolution. As a consequence, often only a part of the representation is displayed at a time and users have to navigate to view the rest.

On the other hand, if the graph gets too large and complex, users have to either deal with representations visualizing only a subset of the data or representations which are cluttered due to too many information. In such situations, interactive adjustments of the representation may become necessary for editing or analyzing certain details. Lenses are versatile tools in this regard, as they allow enriching, suppressing or altering content in a user-defined region of interest of the visual representation.

In this chapter, solutions for supporting the user concerning both issues are presented. In section 6.1, a novel approach supporting the exploratory analysis of graphs with navigation recommendations to interesting data subsets is introduced. Section 6.2 provides a comprehensive overview of the large variety of existing lens techniques and a conceptual model for lenses. The chapter is summarized in Section 6.3.

Contents

6.1. Navigation Recommendations for Exploring Hierarchical Graphs 117

6.1.1. Problem Analysis 117

6.1.2. Approach 120

6.1.3. Implementation 126

6.1.4. Use Case 127

6.1.5. Discussion 128

6.2. A Comprehensive Overview of Lenses 129

6.2.1. Problem Analysis 130

6.2.2. Definition of Interactive Lenses in Visualization 130

6.2.3. Categorization of Lenses according to Data and Tasks 138

6.2.4. Lens Interaction and Display 147

6.2.5. A Demonstrating Example: the *EditLens* 149

6.2.6. Discussion 151

6.3. Summary 152

Major parts of the approach introduced in Section 6.1 appeared in the following publication:

[GST13] S. Gladisch, H. Schumann, and C. Tominski. "Navigation Recommendations for Exploring Hierarchical Graphs". In: *Advances in Visual Computing, Proceedings of the International Symposium on Visual Computing (ISVC)*. Ed. by G. Bebis, R. Boyle, B. Parvin, D. Koracin, B. Li, F. Porikli, V. Zordan, J. Klosowski, S. Coquillart, X. Luo, M. Chen, and D. Gotz. Vol. 8034. Springer, 2013, pp. 36–47.

- The author of this thesis developed and implemented the concept of this approach and wrote major parts of this publication.

Section 6.2 is based on the following state-of-the-art report and an extended version thereof, which was submitted to the journal *Computer Graphics Forum*.

[Tom+14] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann. "A Survey on Interactive Lenses in Visualization". In: *EuroVis State-of-the-Art Reports*. Eurographics Association, 2014, pp. 43–62.

- The author of this thesis is mainly responsible for the description of the lens properties and the categorization of lenses according to tasks and data types.

For this chapter, the published work has been adjusted and extended in the following way:

- The wording was adjusted to match the wording of this thesis.
- The problem analysis from Section 6.1.1 was extended to emphasize the connection of the proposed approach to visual graph editing. In addition, the examples were replaced.
- The major steps of the approach described in Section 6.1.2 were revised.
- Section 6.1.5 further introduces a generalization of the proposed approach.
- The conceptual model of lenses was described on the basis of the more comprehensive data-state-reference model [CR98] instead of the visualization pipeline [CMS99].
- The categorization of lenses according to data and tasks was reformulated to focus on graphs and tasks of visual graph editing.
- Section 6.2.4 was shortened and newly published work concerning lens interaction and display was included.
- The detailed lens example from Section 6.2.5 was replaced with the *EditLens* from Section 4.1.

6.1. Navigation Recommendations for Supporting Exploratory Analysis of Hierarchical Graphs

Within the scope of this thesis, a novel approach for providing navigation recommendations to interesting data in hierarchical graphs was developed. It addresses the exploratory analysis of hierarchical graphs and aims at facilitating the user's navigation decisions (i.e., recommend navigation to interesting targets) and mitigating the trial-and-error character of navigation (i.e., reduce unconscious navigation through regions with uninteresting data). After defining the problems to be solved and briefly reviewing related work in the following section, the approach is described in detail.

6.1.1. Problem Analysis

Graph analysis can be an important part of visual graph editing. On the one hand, exploring unknown graphs can lead to the discovery of erroneous or missing data that need to be edited. On the other hand, analyzing a graph after applying certain edit operations can provide insight about the editing effect. Analyzing unknown graphs almost always includes switching between overview and detail representations and navigating to different parts of the graph. These tasks are typically supported by a zoomable representation and a hierarchical abstraction of the graph[1]. The zoomable representation enables the user to pan to any rectangular partial view of the graph layout. This is called *horizontal* navigation hereafter. The hierarchical abstraction allows for additional *vertical* navigation: the user can expand or collapse nodes in order to get to a lower or higher level of abstraction [EF10]. Figure 6.1 illustrates both types of navigation.

There are several existing systems that implement both navigation strategies [BHJ09; TAS09; Aub04]. An advantage of these systems is that users can freely choose the part of the data they are interested in and the level of abstraction that suits their needs.

However, a problem is that users may be overwhelmed with the seemingly infinite number of possibilities for navigation. According to Spence [Spe07], a key question for the user is: *Where should I go now?* Figure 6.2 illustrates this problem. Considering a current position arrived at during an exploratory analysis, the user does not know where interesting data could be located. Shall the user pan in this direction or the other to find some low-degree nodes that could indicate missing edges? Which node should I expand to uncover nodes with abnormal attribute values that might be erroneous? Although the user can derive some more or less vague answers from the visual representation itself (e.g., navigate to where many edges connect), finding interesting data in an unknown graph often leads to a tedious trial-and-error navigation procedure. For these reasons, a dedicated support to assist the user during navigation would be a promising addition to the user's analytical toolbox. Existing solutions in this respect can be divided into two

[1]The hierarchical abstraction of a graph leads to a *hierarchical graph* which is defined as a rooted tree whose leaves correspond to a graph at the finest level of granularity. Inner nodes of the tree correspond to aggregations or abstractions of their associated child nodes [HMM00].

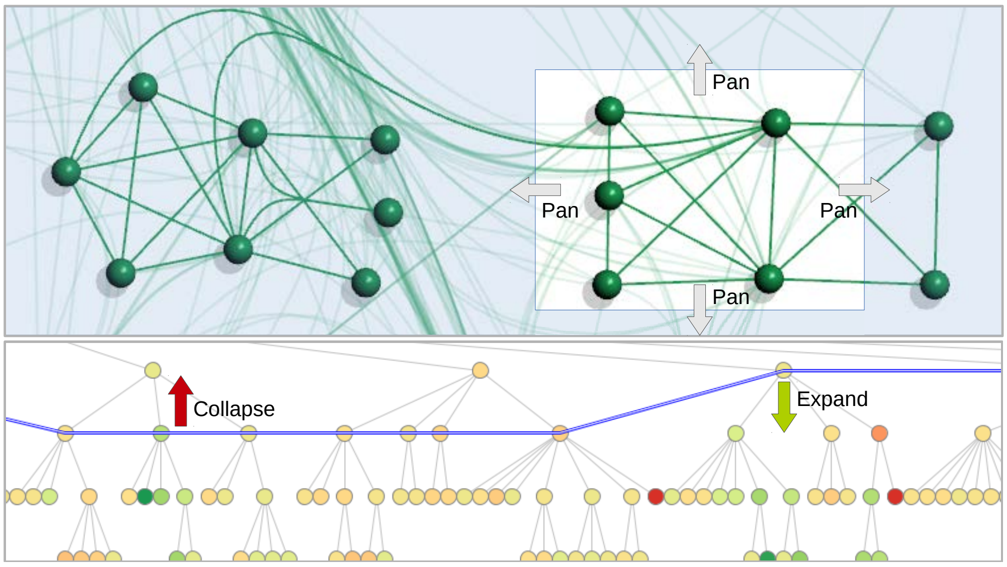


Figure 6.1.: Navigation in a hierarchical graph. Horizontal navigation means altering the partial view on the graph layout (e.g., by panning the view). Vertical navigation means adjusting the level of abstraction (blue line) along the graph hierarchy by expanding or collapsing individual nodes. (Colors visualize data attributes associated with nodes.)

categories. On the one hand, there are approaches that focus on providing *orientation help* to keep users oriented. On the other hand, *navigation recommendation* approaches aim to actually suggest navigation steps to the user. In the following, important examples are briefly reviewed.

Orientation help This kind of assistance helps users to orient themselves while navigating through the data. In the context of graph exploration, May et al. [MSK12] present a technique that computes landmarks in the context of the current partial view (focus). The visualization is enhanced with labeled signposts that show directions to the determined landmarks. Jusufi et al. [Jus+12] investigate orientation guidance in graphs for which a complete partition is given. The approach is based on special glyphs that provide overviews of certain subgraphs connected to a focus node. Plaisant et al. [PGB02] also use special glyphs for user orientation. They enhance a tree visualization with preview icons that summarize the topology of subtrees. From a more general perspective, off-screen visualization techniques (e.g., [FD13], [Gus+08], [BR03]) can also be considered to be an orientation help.

Navigation recommendations The main idea of navigation recommendations is to suggest navigation steps (e.g., a specific target or a direction). Van Ham and Perer [HP09] describe an exploration model for graphs that includes navigation recommendations. Based on an initial focus, the approach is to compute and show the most interesting

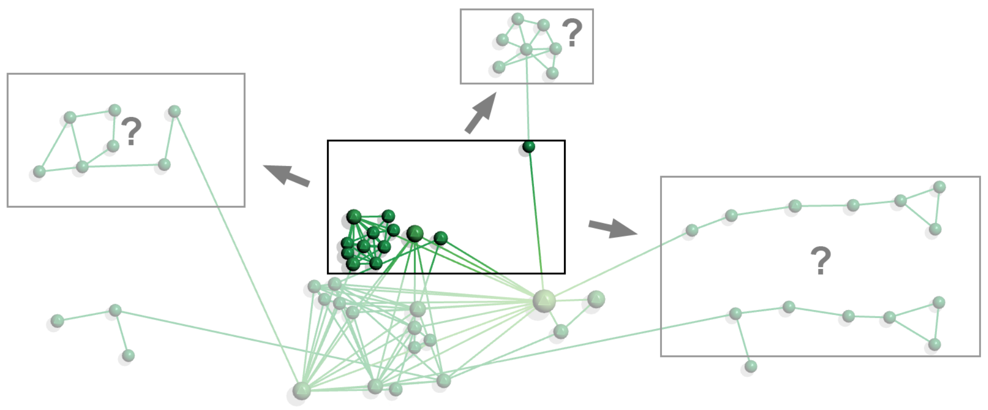


Figure 6.2.: The problem of navigation. Given a partial view on a graph layout (center rectangle) the user does not know where to navigate in order to find "interesting" data (rectangles with question mark).

contexts. Visual hints help users to decide which nodes in the context to expand in order to navigate to the additional information. Crnovrsanin et al. [Crn+11] present a technique that recommends interesting nodes based on a set of selected nodes. Interestingness of nodes depends on data attributes, graph structure, and sequences of previous user interaction. Perer and Van Ham [PH11] introduce *querying and browsing* as a new paradigm for graph exploration. They propose a general model that determines an initial focus and its context on the graph based on a textual query. Special icons within a node-link representation recommend to the user where to browse the context in order to find interesting information. Additionally, the approach computes and visualizes the shortest path from a focus node to a recommended node in the context.

Open research questions The reviewed examples from the literature demonstrate how useful user assistance can be. A detailed look into the mechanisms behind the existing solutions reveals that most of them define a notion of a current *focus* that is associated with a *context*, where focus and context are defined exclusively on the graph structure. However, this implies that navigation recommendation can be given only for entities being connected to the focus in terms of the graph's structure. Interesting but disconnected nodes (e.g., in graphs with disconnected components) cannot be recommended, even if they are located close to the focus in the graph's layout (which is what users see on the display). The novel approach presented in the following section addresses this limitation by using a broader and more general notion of focus and associated context.

Another aspect common to the reviewed solutions is that they address only horizontal navigation in plain graphs. Hierarchical graphs have not been considered in connection with navigation recommendations so far. In this thesis, this issue is tackled by including vertical navigation along the axis of the level of abstraction. In other words, the novel approach addresses horizontal navigation *and* vertical navigation.

6.1.2. Approach

The novel approach for providing navigation recommendations addresses zoomable node-link representations of hierarchical graphs, where attributes can be associated with nodes. As it aims to support the user in deciding which navigation step to take next for arriving at interesting data, the following key issues must be addressed:

Determining recommendation candidates Given a hierarchical graph and the current state of the visual exploration process, a set of recommendation *candidates* has to be determined.

Specifying and detecting interesting recommendations In order to compile a set of navigation *recommendations*, the candidates' *interestingness* must be specified and those that are worth visiting next must be detected.

Communicating recommendations visually The selected navigation recommendations need to be *communicated* to the user in an unobtrusive fashion with as little distraction from the actual visualization as possible.

Following this idea, more details about how the novel approach handles these issues are presented below. But first of all, possible targets for navigation recommendations need to be defined. In general, a system could recommend navigation to any entity related to a graph, for example, nodes, edges, connected components, cliques, or any other semantically meaningful subset of nodes and edges. For the sake of simplicity, here the considerations are restricted to nodes as the targets for navigation recommendations.

6.1.2.1. Determining recommendation candidates

As commonly accepted, the starting point for determining candidates is the user's current focus. Based on the focus, a context is defined which contains the candidates. The context must include a sufficiently large number of candidates to choose from, and it must be sufficiently small to stay focused and to avoid computations on a huge search space. The size of the context and hence the number of candidates is controlled by means of a distance measure. In summary, three components are utilized: (1) a set of focus nodes to start with, (2) a sufficiently sized set of context nodes – the candidates, and (3) a distance measure to control the size of the context. Figure 6.3 illustrates how these components can be realized.

An intuitive and often used definition of these components is based on the graph's structure. A set of focus nodes is selected by the user, and the context is defined by the k-neighborhood[2] of the focus nodes. Here the parameter k can be adjusted to control the size of the context.

As already indicated, focus and context are generalized to a broader definition in this thesis. So, as a second facet, the user's current view on the graph layout is additionally

[2]The *k-neighborhood* of a node v is the set of nodes $N_k(v)$ whereas for each node $v' \in N_k(v)$ exists a path between v and v' with maximal length k.

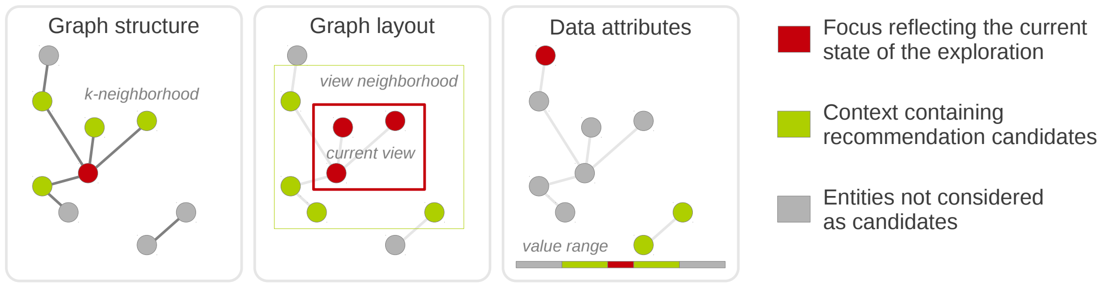

Graph structure
k-neighborhood
Graph layout
view neighborhood
current view
Data attributes
value range
Focus reflecting the current state of the exploration
Context containing recommendation candidates
Entities not considered as candidates

Figure 6.3.: Different definitions of focus and context in terms of the graph structure, the graph layout, and the data attributes yield different candidates for recommendation.

considered. That is, all nodes that are currently visible on the display are considered to be the focus. Again, the context is defined in terms of a neighborhood, but this time a neighborhood in the view space. The size of the context is again controllable by a distance measure, for instance the *euclidean distance*.

So far, data attributes that might be associated with the nodes have not yet been taken into account. Consequently, a focus is also allowed in attribute space. This focus can be determined in different ways, for example by dynamic filtering or by fixing the value range of what is currently visible on the display. The context can then be defined as a range of values enclosing the focus, where the range's size can be set as needed.

With this general definition the different aspects being relevant when exploring graphs – the graph structure, the graph layout, and the data attributes – are captured. The broader definition also allows to circumvent problems that occur when considering one of the aspects alone. An example are nodes that are close to the focus in the layout, but that are far away in terms of the graph structure. In contrast to existing solutions, the approach presented in this thesis is able to recommend navigation to such nodes.

6.1.2.2. Specification and Detection of interesting recommendations

With the definition of candidates in the context, the next step is to assign an interestingness to each candidate. As the aim is to recommend interesting nodes to navigate to, a concept is needed that specifies how interesting a recommendation candidate is. Given the unpredictability of the visual exploration process, the concept must be capable of handling varying interestingness.

An established concept in this respect is the *degree of interest* (*DOI*), a numerical interestingness computed by means of a *DOI* function. Originally, FURNAS [Fur86] introduced the *DOI* for trees and computed the interestingness of a node x concerning its a priori interest (*API(x)*) and its distance to a user-selected focus node y (*DIST(x,y)*):

$$DOI(x,y) = API(x) - DIST(x,y)$$

As a concrete definitions for *API(x)*, the nodes distance to the tree's root node is suggested. For *DIST(x,y)*, the path length from x to y in the tree is used.

Van Ham and Perer [HP09] adapted the *DOI* for graphs. For *API(x)* a function based on a structural property (e.g., the node's degree) or a node attribute value of x is used. As a plausible definition for *DIST(x,y)*, the length of the shortest path between x and y is proposed. To additionally capture the user's interest in certain properties or attribute values of nodes, a third component *UI* is added. This leads to

$$DOI(x,y) = \alpha \cdot API(x) + \beta \cdot UI(x) + \gamma \cdot DIST(x,y)$$

whereas α, β, γ are real-valued weights to be able to prioritize certain components.

In context of the analysis of large dynamic networks, Abello et al. [Abe+14] generalized the concept of *DOI* again. Besides structural properties and attribute values of nodes and edges, they also consider their temporal evolution. Moreover, instead of a static *DOI* function, a modular *DOI* function, which can be interactively assembled from predefined functional components, is introduced. This however requires interaction with a complex graphical user interface. A generalization is the use of a set of focus nodes instead of a single focus node.

In this thesis, a *DOI* function is used that is capable of computing an interestingness concerning a multi-focus $F_* \subseteq \{F_{struct}, F_{view}, F_{attr}\}$, whereas F_{struct} is a set of user-selected focus nodes (i.e., focus on the graph's structure), F_{view} is the user's current view (i.e., focus in view space) and F_{attr} is a set of user-selected attribute value ranges (i.e., focus in attribute space). Note, while F_{view} is automatically defined and thus always element of F_*, F_{struct} and F_{attr} must be selected by the user before they are included. The *DOI* function is specified as follows:

$$DOI(x, F_*) = \alpha \cdot API(x) + \beta \cdot UI(x) + \gamma \cdot DIST(x, F_*) + \delta \cdot KNOW(x)$$

As can be seen, the *DOI* function computes the interestingness of a node x according to an additional weighted component *KNOW*. This component captures a node's interestingness according its exploration state, i.e. if a node has already been visited (i.e., has been visible or has explicitly been marked as explored) in the course of the exploration. This allows to either penalize already explored data or, on the contrary, favor them. Which option to use depends on the user's goal. Visited nodes can be considered less interesting because they do not provide any new information. On the other hand, they could be particularly of interest for comparison tasks.

Given the above-described specification of the *DOI* function, an open question is how to instantiate its components and weights. This approach follows the accepted way of previous *DOI*-related approaches in this regard and provides interactive means for the user to define and adjust the components and their weights. To ease the definition procedure, a set of selectable template functions with identical value range $[0,1]$ is provided for every component. For *API* and *UI*, template functions based on node properties or attribute values are provided (e.g., the normalized node degree or the normalized value concerning a specific node attribute). For *DIST*, template functions to compute a combined distance to every existing focus (e.g., minimum, maximum and average) as well as means to steer the size of every focus' context (e.g., a slider for defining k of the k-neighborhood of the focus on the graph's structure) are offered. To instantiate the $KNOW$ component, template functions based on the exploration state are made available (e.g., a function that outputs 0 or 1 if the node has already been visited or not). Which template functions to select (i.e., what is interesting?) and how to parameterize them depends on the application domain, the use case, and the analyzed graph. To enable users to prioritize certain components of the DOI function, it is suggested to provide interactive means to select α, β, γ and δ within $[0, 1]$.

The described DOI function is a trade-off between flexibility of describing a node's interestingness and interaction effort for instantiating it. This approach allows a sufficient DOI definition,but the approach from ABELLO ET AL. is more flexible [Abe+14]. However, this comes at the cost of higher interaction effort to define the DOI function.

Given an appropriate instantiation of a DOI function, the interestingness of the graph's nodes is computed. It is worth mentioning that this is exclusively done for the recommendation candidates in the context of the current focus. This spares computing interestingness values for all nodes of the whole dataset.

For a hierarchical graph, two alternative ways of computing the interestingness are differentiated. First, the interestingness of nodes on the finest level of granularity can be computed and then can be aggregated along the hierarchy. The second alternative is to compute interestingness explicitly for each candidate irrespective of whether it is a leave node or an inner node. Again, the application scenario and the nature of the data decide on which alternative to apply.

Now that every candidate has a DOI value, the candidates are sorted according to their interestingness. The result is a ranking of the recommendation candidates. To detect the *most interesting* nodes which are to be recommended, the first m nodes of the ranking are selected as targets for the navigation recommendations. Thereby, m should be kept small to avoid overwhelming the user with too many recommendations. Experiments within this thesis showed that recommending $m < 10$ interesting navigation targets is sufficient.

6.1.2.3. Communicating recommendations visually

The last step is to create an adequate visualization for the navigation recommendations. Given a visually rich graph representation, it is a question how it can be enhanced in order to communicate navigation recommendations to the users without interfering with the ongoing visual exploration. In this work, an answer to this question is to embed specifically designed visual navigation cues into the existing node-link representation. Depending on the type of navigation and on where the target of a recommendation is located, different visual cues are utilized. The type of navigation can be either *horizontal* or *vertical*.

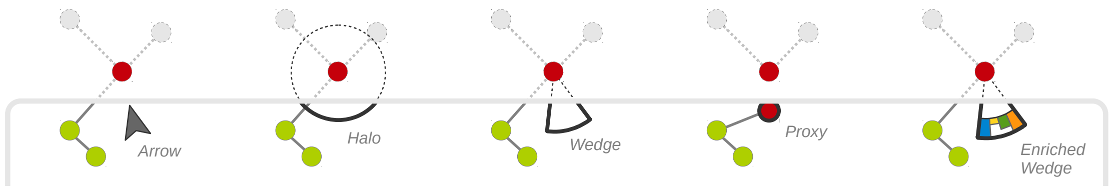


Figure 6.4.: Techniques for recommending navigation to off-screen nodes. Green nodes are on-screen, dashed elements are off-screen, and the red node is the recommend target.

Recommendation for horizontal navigation For horizontal navigation, it is distinguished whether nodes are *on-screen* or *off-screen*. Recommendations to on-screen nodes are visualized via subtle highlighting rings that encode how interesting a node is according to its *DOI* value.

For recommendations to off-screen nodes, a visual encoding is needed that communicates at least the target's direction and preferable also the distance to the target. For this purpose, known techniques for off-screen visualization are considered, such as arrows, *halos* [BR03], *wedges* [Gus+08], or *proxies* [FD13]. Arrows are easy to interpret, but communicate navigation direction only. Halos and wedges have the advantage that they encode direction and distance to a recommended navigation target. Further, wedges can be arranged to reduce overlap [Gus+08]. Proxies usually focus on communicating additional information about the target by means of shape, color, or labels instead of its distance.

Inspired by these approaches, a new solution was designed that combines the advantages of the existing ones. So-called *enriched wedges* are visual cues that encode direction, distance and also additional information about *why* the recommendation was given. This is accomplished by embedding a bar chart into a wedge. The wedge visualizes direction and distance, and each bar visualizes the partial interestingness of a recommended node according to the individual components of the *DOI* function (i.e., *API*, *UI*, *DIST*, *KNOW*). Using enriched wedges can positively influence the user in arriving at a navigation decision (i.e., choosing the right navigation target). Figure 6.4 illustrates the enriched wedge in comparison to existing off-screen techniques.

Note that enriching the navigation recommendations with a visualization of values of individual *DOI* components is not restricted to wedges pointing to off-screen targets. The highlighting of on-screen targets can be enhanced in a similar manner to provide information about interestingness at a glance.

Recommendation for vertical navigation A vertical navigation is necessary when the recommended target is not contained in the currently visualized level of abstraction. As the target is definitely not visible, a suitable anchor must be picked to attach the navigation recommendation to. It was decided to visually highlight the nodes whose expansion (or collapse) would either bring the recommended target to the display, or closer in terms of the graph hierarchy. For example, if a target is below the current level

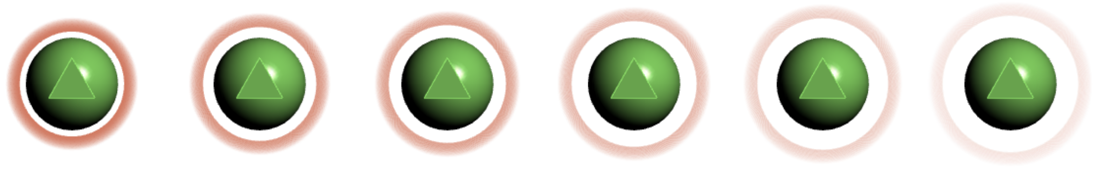


Figure 6.5.: Snapshots of the animation that indicates nodes to be expanded to arrive at a recommended navigation target.

of abstraction, the target's ancestor that is contained in the current level of abstraction is highlighted. If the ancestor is off-screen, the off-screen techniques described before can be applied again.

In order to differentiate the highlighting for vertical navigation from that for horizontal navigation, and further the one for node expansion from that of node collapse, animated rings around nodes are used. In accordance with the goal to generate an unobtrusive visual embedding, the highlighting is designed as subtly pulsing animations with a specific direction. The animated rings appear to shrink when collapse navigation is recommended and to grow for recommended expansion. Figure 6.5 shows snapshots of an animation indicating an expand recommendation.

6.1.2.4. Summary and additional concerns

A novel data-driven approach for navigation recommendations to interesting information was introduced. The proposed solution is based on three basic steps: (i) collection of a set of recommendation candidates based on a compound focus, (ii) specification of the user's interest and detection of the most interesting candidates, and (iii) visualization of navigation recommendations via visual cues embedded into an existing graph representation.

A key issue of the described approach is balancing it appropriately. Interests vary and also the visual presence of navigation cues will be perceived differently by different users in different stages of the exploration process. Therefore, it is critical to adjust the computation of recommendations and their visualization to the application scenario and to the preferences of the user. The novel approach provides the required flexibility to do so. Further, the on-demand character of the proposed approach is recalled. Only if users feel that they need assistance they will activate the navigation recommendations.

Two additional issues need to be addressed in the context of navigational guidance: (i) a good initial view to start with and (ii) a visual encoding of the exploration state. Both are not trivial questions and have not been dealt with in depth in this work. Yet some ideas how to address them have been developed.

Ideally, a good initial view on the data provides an expressive overview of the data and offers a suitable number of options for further exploration. For determining such an initial view, different criteria can matter. For example, the number of nodes can be considered. Huang et al. [HEW98] state that 20 to 100 nodes are suitable for an overview. Moreover, in specific applications, there may exist data elements being semantically more relevant than others. In such cases, including graph elements of higher relevance (e.g., outliers) can lead to a more appropriate initial view. One could also favor nodes with high degree as they potentially lead to more options for navigation along the graph structure. Despite these initial suggestions, creating a good initial view remains a difficult and context-dependent task.

The second concern regards the dependency of interestingness on the exploration state. To make this dependency clear to the user it makes sense to visualize a node's exploration state as well, because it may influence navigation decisions. When exploring hierarchical graphs the user might want to know which subtrees have already been explored. Additionally visualizing this information is difficult. Therefore, experiments with an on-demand labeling were conducted that classifies nodes into *unexplored*, *partially explored*, and *explored*. Such on-demand labels can help users to decide where to explore further and where no further exploration is necessary. Another recent approach utilizes a heat map highlighting of nodes for this purpose [ZK15; ZK14].

6.1.3. Implementation

To test the proposed approach, a proof-of-concept implementation was prepared. As a basis for computing the underlying zoomable node-link representation of a hierarchical graph, *CGV* was utilized [TAS09]. *CGV* is a graph visualization system whose particular emphasis is on interaction. It provides several interlinked views which can be interacted with in various ways. This includes dynamic filtering, graph lenses and techniques for horizontal and vertical navigation.

A plausible default preset for the definition of the *DOI* function (including maximum attribute values and attribute outliers) was implemented, which allows to give recommendations at all times, even in cases where the user has not yet made her interests known to the system. The definition of components as well as weights of the *DOI* function can be altered interactively via a simple graphical user interface. Interesting off-screen nodes are visualized by *enriched wedges*. Moreover, depending on the data characteristics and the application scenario, it can be selected whether *DOI* values are aggregated along the hierarchy or computed individually for every candidate irrespective of its hierarchy level. The current implementation is able to recommend nodes in the context of a selected set of focus nodes and the view context of the current view. The suggested third focus in attribute space is yet to be included.

The system was tested on a hierarchical graph with 695 nodes and 4073 edges where nodes represent search queries and edges depict relations between queries. Figure 6.6 shows a partial view on the graph a user might see during exploration. This figure shows recommendations to investigate on-screen targets, indicated by red circles around some nodes of the context of the yellow encircled focus node. An expansion recommendation indicated by a grow animation around the focus node communicates that there is at least one more interesting target below the current level of abstraction. Enriched wedges at the border of the screen recommend navigation to off-screen targets in the context of

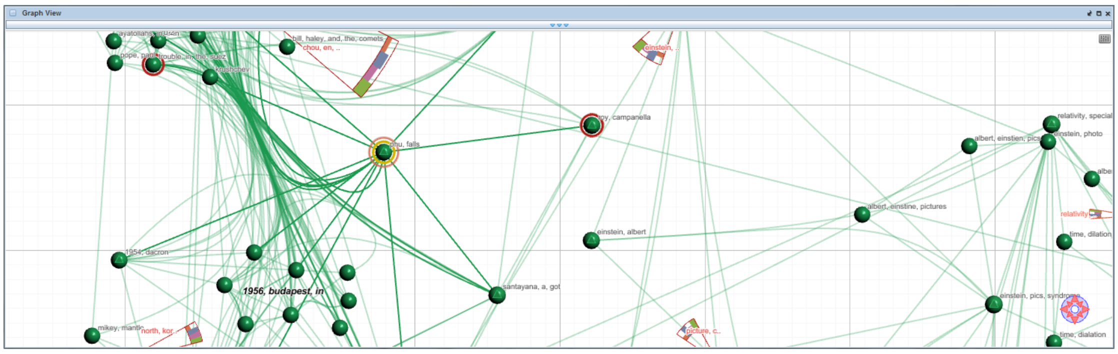


Figure 6.6.: Different kinds of navigation recommendations in the proof-of-concept implementation. Deep red circles indicate on-screen targets within the context of the currently selected focus node (highlighted with a yellow circle) which are worth investigating next. The red circle around the focus nodes belongs to a grow animation and thus recommends node expansion. Enriched wedges indicate off-screen targets in the structural context of the selected focus node (e.g., node "chou, en, ..") and the view context (e.g., node "picture, c..") that might be of interest to the user as well.

the focus node and the view context. A recommended target which is not included in the structural context of the focus node is the node "picture, c.." at the bottom center for example. Such a navigation recommendation cannot be provided by any approach in literature.

Once the user has decided on the next navigation target, it can be visited by following the navigation recommendations manually. For example, the user can pan the view in the direction of an enriched wedge until the target falls into view. The target is then highlighted using the red circle for on-screen targets. As an alternative to manual navigation, *CGV's* animation facilities are utilized to provide automatic animated traveling to the selected target. To this end, the user simply clicks an enriched wedge to trigger the animation.

During the exploration, the recommendations are constantly updated according to the current focus and the specification of the user's interest. The system also keeps track of which nodes have been explicitly marked as explored.

In summary, provided that users are able to express their current interest, this implementation helps in locating interesting nodes in the local context quickly.

6.1.4. Use Case

Right from the beginning it is often not clear, whether and if so which data aspects of a graph must be edited. Especially if the graph is unknown, the user must obtain information about the data first to deduce the need for editing. This can happen in the course of an exploratory analysis based on graph visualization. The same applies for scenarios in which information about the effect of previously performed edit operations must be received. As already described, the exploration of a graph almost always includes navigating to different parts of the data. For supporting the user in situations where it is difficult to decide the next navigational steps, the proposed approach can be used. The automatically provided navigation recommendations show where interesting data is located. Following the navigation recommendations may help to reveal the need for editing the graph or to obtain information about the effect of a previously applied edit operation. However, this substantially depends on whether the user is able to express her interest concerning this matter in a certain situation.

An example is the exploration of a large and strongly connected graph where the user recognizes a node which appears strange on the first sight as it is connected to one node only. Now, the user wants to figure out whether this node is an exceptional case (which might even be erroneous and must be edited), or whether similar nodes exists in other parts of the graph, which have not been explored yet. As the user is overwhelmed by the large number of available navigation possibilities and has no idea where such low-degree nodes could be located, the user decides to get assistance via navigation recommendations. For that, the system is activated and the current interest is defined. Thereupon, navigation recommendations to low-degree nodes located in a specific part of the graph outside the current view are given. This eases the user's decision where to proceed the exploration. Trial-and-error navigation is not necessary.

6.1.5. Discussion

The proposed approach extends the body of existing work with regard to the following aspects. First, hierarchical graphs were addressed with horizontal and vertical navigation, an issue which has not been studied in previous work. Second, for specifying the user's interest, the widely-accepted *DOI* concept was extended by the component *KNOW* which captures the exploration state of data elements. Further, in terms of detecting recommendation candidates, the notion of focus and context was generalized to incorporate the graph's structure, the graph's layout, and the graph's attributes. And last but not least, for communicating navigation recommendations, several visual encodings were suggested, including the novel *enriched wedge*. As can be seen, the proposed approach extends and generalizes the key steps which are necessary to compute navigation recommendations.

The whole approach can be further generalized to the concept of event-based visualization presented in [Tom06]. The basic idea of this concept is to detect "[...] special portions of the data that are of particular interest [...]" [Tom06] (*events*) which are then automatically highlighted in the visual representation. The steps of the proposed approach perfectly match the three basic steps of event-based visualization – event specification, event detection and event representation – as described in the following:

1. Specification of the user's interest $\rightarrow$ event specification: The event specification's task is to compile conditions (*event types*) for deciding whether a data portion is interesting or not. This corresponds to defining the threshold m for selecting the most interesting recommendation candidates according to their *DOI* values.

2. Detection of recommendation candidates → event detection: The event detection is used to find portions of the data which satisfy the compiled conditions of the last step (*event instances*). Thus, this step corresponds to the detection of the m most interesting recommendation candidates.

3. Representation of navigation recommendations → event representation: The event representation's purpose is to make users aware of the fact that event instances have been found and to communicate different event types visually (if existent). This encompasses the representation of navigation recommendations with visual cues and highlightings, which are considered as *implicit event representations* by TOMINSKI[3].

This generalization opens up many possibilities to adjust individual steps of the proposed approach. Examples include alternative conditions to specify the user's interest in the data and alternative representations of navigation recommendations, for instance an *explicit representation* in separate views.

The usefulness of the proposed approach is indicated by a working proof-of-concept implementation. However, in the future, alternative interfaces for specifying the users' interests should be investigated. The current approach for defining the *DOI* function needs to be revised in order to make it more flexible for users. For that, the solution provided in [Abe+14] should be considered. Further it makes sense to keep a history of what users have already marked as interesting. This would allow to find better starting points for exploration.

An urgent issue is that the degree to which this approach reduces the trial-and-error character of visual exploration has not yet been quantified in user studies. Due to the many influencing factors, this is a challenging task and clearly requires the expertise of evaluation specialists. For this reason, the evaluation of the proposed approach is left for future work.

6.2. A Comprehensive Overview of Lenses

Whereas the previous section addressed the issue that users have to navigate in order to view different parts of a graph representation, this section addresses representations with too few or too many information hindering the editing or analysis of details. To better support the user in this regard, the concept of lenses is investigated, which allows to locally enrich a visual representation with details or to filter out irrelevant information. As a large variety of lens techniques already exist, this sections aims at providing a systematic overview and conceptual model of lenses. This is helpful for editing and analyzing graphs with regard to two aspects. First, a systematic overview of lenses can

[3]In [Tom06], TOMINSKI distinguishes between *implicit event representation* and *explicit event representation*. Implicit event representation means, that events are visualized within the original visual representation of the data. Explicit event representation concerns the separate visualization of events.

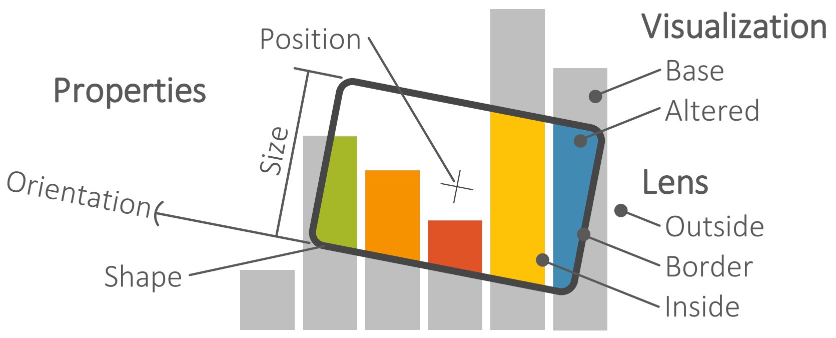


Figure 6.7.: Schematic depiction of an interactive lens.

assist users in selecting existing lens techniques for supporting certain tasks. Second, a conceptual model of lenses describing the possibilities of lenses in general also facilitates the development of novel lens techniques dedicated to graph analysis and editing.

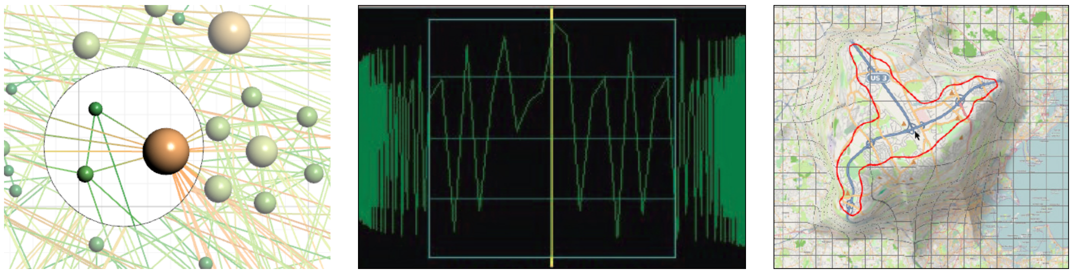
6.2.1. Problem Analysis

When analyzing or editing a large and complex graph visually, situations may occur where the representation provides too few or too many details for carrying out the task at hand. For example, a matrix representation might not encode a specific edge attribute the user is currently interested in, or a node-link representation might suffer from edge clutter, which hinders identifying individual edges. To alleviate such issues, the representation must either be enriched with necessary details, or unnecessary information must be filtered out. As interactive lens techniques provide powerful and versatile means for doing so, they are considered in the following.

The basic idea behind interactive lenses is to provide an alternative visual representation of the data underlying a user-defined local area of the screen on demand. In early works on *magic lenses*, Bier et al. envisioned several possible instantiations of useful lenses, including lenses for magnifying visual items, adjusting graphical properties, querying precise data values, or dynamic filtering [BSP97; FS95; SFB94; Bie+94; Bie+93]. Nowadays, more than 40 lens techniques dedicated to the closer context of visualization research exist. Still more can be found in related fields that deal with visual information as well (e.g., human-computer interaction or augmented reality). However, a comprehensive overview as well as a general model of lenses is still missing. Before providing both, a definition of interactive lenses in visualization is given first.

6.2.2. Definition of Interactive Lenses in Visualization

As illustrated in Figure 6.7, a lens can be defined as a selection according to which a base representation is altered. Which part of the base representation is selected can be steered interactively by adjusting the *lens properties* shape, position, size and orientation.



(a) Circular shape (b) Rectangular shape (c) Content-adaptive shape

Figure 6.8.: Lenses with different shapes. (a) Circular *Local Edge Lens* [Tom+06], (b) Rectangular *SignalLens* [Kin10], (c) Content-adaptive *JellyLens* [Pin+12].

Originally, the visible effect of a lens was locally confined to within the lens [Bie+93], in contrast to overview+detail and focus+context techniques that affect the display globally [LA94]. Yet, in the context of visualization there are less strict interpretations of what a lens is. A lens technique might affect the visualization beyond the confines of the lens or even show the lens and its effect separately. These techniques will be considered as well as long as they have a lens-like feel or call themselves lenses.

Next in Section 6.2.2.1, general properties of lenses are investigated first. Thereafter, in Section 6.2.2.2, a conceptual model of lenses in the visualization domain is introduced.

6.2.2.1. Properties of Lenses

From a user perspective, the lens geometry is important as it generally determines the selection. In other words, the lens geometry defines where a lens takes effect or at least which data entities are affected (e.g., which details are additionally presented or filtered out). The key properties to look at are: the lens shape, the position and size of a lens as well as the orientation of a lens.

Shape The most prominent property of a lens is its shape. In principal, there are no restrictions regarding the shape of a lens, but it is usually chosen to meet the requirements of the application and to go hand in hand with the lens' effect. Following the classic prototype of real-world lenses, many lenses are of circular shape. An example is the *Local Edge Lens* [Tom+06] as shown in Figure 6.8(a). Yet, in a world of rectangular windows on computer screens, it is not surprising to find lenses that are of rectangular shape, such as the *SignalLens* [Kin10] in Figure 6.8(b).

An interesting alternative are lenses that are capable of adapting their shape automatically according to the characteristics of the data. Such self-adapting lenses are particularly useful in cases where the lens effect needs to be confined to geometrically complicated features in the visualization. Examples are the *smart lenses* [TFS08a], which adjust themselves to shapes of geographic regions, or the *JellyLens* [Pin+12], which morphs around arbitrary geometric features in the data (see Figure 6.8(c)).

While two-dimensional shapes are prevalent, there are also lenses with 3D shapes. They are typically used for the visualization of 3D data or 3D graphics in general. Examples are lenses for flow visualization [FG98], for the magnification of volume data [LHJ01], lenses for 3D scenes [RH04; CCF97], or bendable virtual lenses [LGB07].

Position and Size The flexibility of a lens in terms of where the visualization is to be altered is defined by the lens position and size. Virtually all lens techniques provide means to adjust these two properties. The position of the lens can be altered to focus on different details of the data, whereas scaling a lens shape up and down controls the extent of the lens effect. Moreover, adjusting the size of a lens is useful for keeping costs at a level that warrants interactivity and comprehensibility at all times. This concerns computational costs, i.e., the resources spent for computing the lens effect, but also cognitive costs, i.e., the effort required to make sense of the lens effect.

Complementary to interactive adjustment are methods to set position and size of a lens automatically. The *RouteLens* [Alv+14] for instance automatically adjusts its position according to a route on a map, while being moved. An example of a lens that automatically adjusts its size to cope with expected costs is the *extended excentric labeling* lens [BRL09]. This lens reduces its size when being moved into areas where too many items are present. Reducing the lens size limits the number of items to be labeled, which in turn helps to maintain interactivity and readability.

Orientation In 2D applications like most graph visualizations, lens orientation is indeed less relevant and therefore often neglected. In fact, for 2D circular lenses, orientation does not even make sense. Yet, orientation can be useful for fine-tuning the lens or is even needed to define the selection at all. As illustrated in Figure 6.9(a), the *PaperLens* [SSD09] relies on 3D orientation to define a slice through a virtual volume in the space above a tabletop display. For another example consider the ellipsoid *FlowLens* [Gas+11] in Figure 6.9(b) and how it is oriented to better match with the underlying visualization.

Orientation becomes more important in settings where multiple users look at one shared display from different angles, as it is the case for collaborative work at a tabletop display [Voi+09]. An example is a user who is handing over a lens to a co-worker at the opposite side of the table, which for certain involves re-orienting the lens.

In summary, lenses exhibit several key geometric properties that can be interactively or automatically adjusted. In addition to what is described here, there can be further properties, for example, a depth range for 3D lenses or a drop-off interval for fuzzy lenses. Usually, such properties are interactively adjustable via standard controls or dedicated lens widgets, as presented in [KRD14].

Now, that lenses along with their properties are defined, the next step is to describe the lens effect. For that, a conceptual model of lenses can be established.

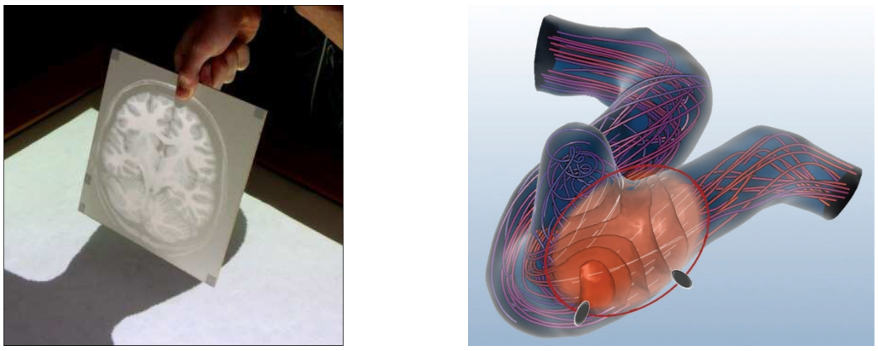

(a) *PaperLens* [SSD09] (b) *FlowLens* [Gas+11]

Figure 6.9.: Differently oriented lenses.

6.2.2.2. Conceptual Model of Lenses in Visualization

To capture the conceptual model behind interactive lenses in visualization, the well-known data-state-reference model [CR98] is taken as a basis. The data-state-reference model describes how data is transformed in a pipeline through the four data stages raw values (V), analytical abstractions (AA), visual abstractions (VA) and image data (I) via *transformation operators* and how the data is transformed at each stage via *stage operators* (see Section 2.2.1). In the following, lenses are considered in the light of these different stages and transformations.

Figure 6.10 shows that a lens can be modeled as an additional lens pipeline consisting of data stages, transformation operators and stage operators that is attached to a *standard pipeline* visualizing a data set. This *lens pipeline* realizes a *lens function* that generates a *lens effect*. There are two points of information exchange between the standard pipeline and the lens pipeline. The first is a *selection* (denoted σ) that defines what is to be processed by the lens function. The second is a *join* (denoted $\bowtie$) that specifies how the result of the lens function is to be integrated back into the standard pipeline.

The Selection σ The selection describes what is to be affected by the lens. In general, the selection is connected to the visual content shown underneath the lens. The user controls the selection by adjusting the lens properties, usually through direct manipulation on the screen [Shn83]. Figure 6.10 illustrates that the lens pipeline is executed only for the subset selected from what is originally processed through the standard pipeline. Often it is assumed that the selection is significantly smaller than the original dataset. This allows a lens to perform calculations that would take too long for the entire dataset.

Defined in image space (I), the lens can be used directly to select a set of pixels on the display. Yet, in principle the selection can be any information that is available along the standard pipeline. In order to do so, the lens needs to be inversely projected from the image space (I) to the model space (VA) in which the geometry and graphical properties

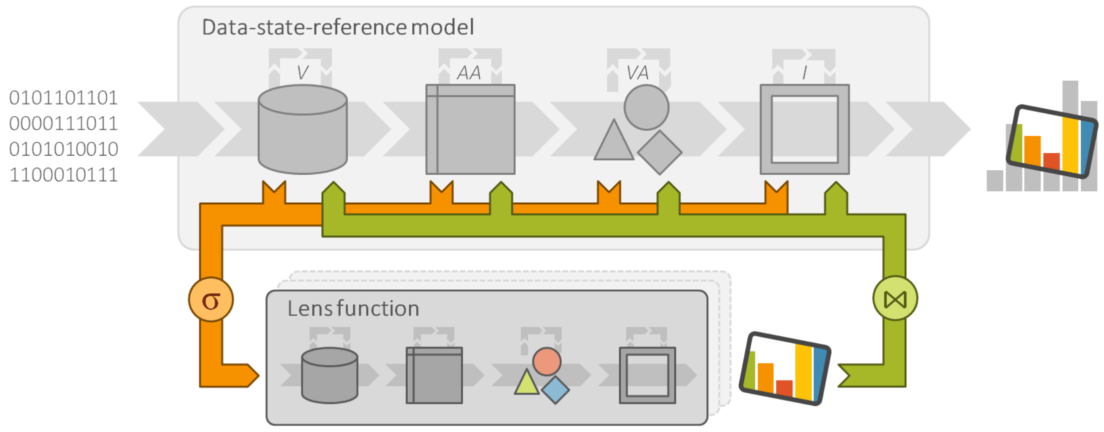
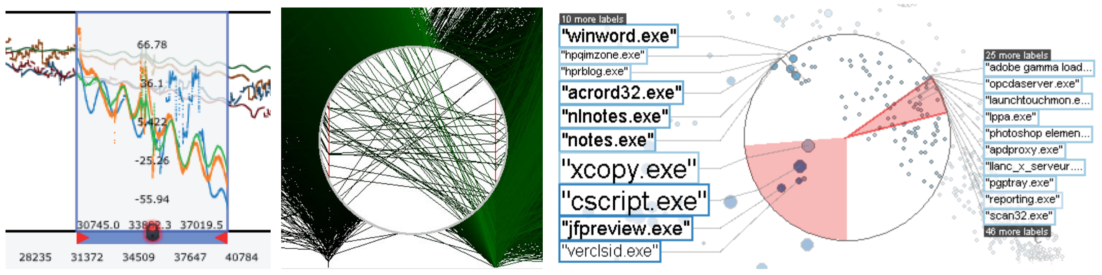


Figure 6.10.: Conceptual model of lenses in visualization. A lens is modeled as an individual visualization pipeline attached to the standard pipeline of the data-state-reference model [CR98].

of the visualization are defined. Further inverse projection to the data space (V or AA) enables selection at the level of data entities or data attribute values. For example, with the *ChronoLenses* [Zha+11] from Figure 6.11(a), the user basically selects an interval on a time scale. The *Local Edge Lens* [Tom+06] from Figure 6.8(a) selects a subset of graph edges that pass through the lens and actually do connect to a graph node within the lens.

So, by appropriate inverse projection of the lens, the selection σ can be made at any stage of the standard pipeline, be it a region of pixels at I, a group of 2D or 3D geometric primitives at VA, a set of data values at V or AA, or a combination thereof. However, what appears simple in theory is not as straightforward in real visualization applications. It can be necessary to restrict the selection to maintain operability of the lens. For example, the *extended excentric labeling* lens [BRL09] from Figure 6.11(c) adjusts itself automatically to keep the selection at a manageable size. The *Layout Lens* [TAS09] from Figure 6.12(a) has to take care not to recursively alter the selection infinitely when the lens function re-arranges nodes on the display. Assigning unique identifiers to data items and maintaining them throughout the visualization process as well as employing the concept of half-spaces can help to properly identify entities to be affected by a lens [TFS08a].

The Lens Function The lens function creates the intended lens effect. Just as any function, so is the lens function characterized by the input it operates on and the output it generates. Obviously, the selection σ is input to the lens function. The lens function further depends on parameters that control the lens effect. Possible parameters are as diverse as there are lens functions. A magnification lens, for example, may expose the magnification factor as a parameter. A filtering lens may be parameterized by thresholds



(a) Alteration (b) Suppression (c) Enrichment

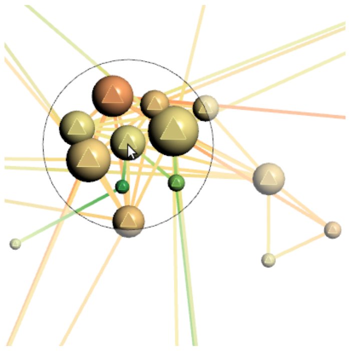
Figure 6.11.: Basic lens functions. (a) *ChronoLenses* [Zha+11] alter existing content, (b) the *Sampling Lens* [ED06b] suppresses content, (c) the *extended excentric labeling* lens [BRL09] enriches with new content.

to control the amount of data to be filtered out. Parameters such as these are essential to the effect generated by a lens. Additional parameters may be available to further fine-tune the lens function. For example, the alpha value used for dimming filtered data could be such an additional parameter.

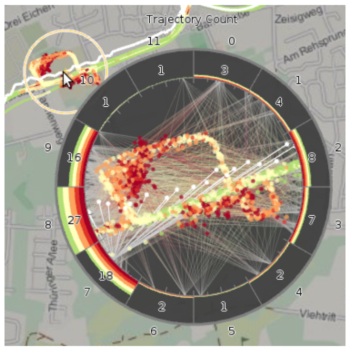
Given the selection σ and parameters, the processing of the lens function typically involves only a subset of the stages of the visualization transformation. For example, when the selection is defined on pixels, the lens function usually manipulates these pixels exclusively at the image stage I. In such a situation, the lens function can be described as a set of stage operators. On the other hand, selecting values directly from the data source V opens up the possibility to process the selected values differently throughout all stages of the pipeline. In this case, the lens function can be described as a set of stage operators and transformation operators, similar to [TFS08a].

The output generated by the lens function is typically an alternative visual representation within the local region defined by the lens. From a conceptual point of view, a lens function can *alter* existing content, *suppress* irrelevant content, or *enrich* with new content, as well as perform combinations thereof. Figure 6.11 illustrates the different options. For example, *ChronoLenses* [Zha+11] transform time series data on-the-fly, that is, they alter existing content. The *Sampling Lens* [ED06b] suppresses data items to de-clutter the visual representation underneath the lens. The *extended excentric labeling* [BRL09] is an example for a lens that enriches a visualization with additional details, in this case with textual labels.

The Join $\bowtie$ Finally, the result obtained via the lens function has to be joined with the base visualization to create the necessary visual feedback. In a narrow sense of a lens, the result generated by the lens function will replace the content in the lens interior as shown for *ChronoLenses* [Zha+11] and the *Sampling Lens* [ED06b] in Figures 6.11(a) and 6.11(b). For many other lenses the visual effect manifests exclusively in the lens interior.



(a) Layout Lens



(b) Time lens

Figure 6.12.: Lens effects beyond the confines of the lens interior. (a) Adjusting layout positions of graph nodes via the *layout lens* [TAS09] also affects the edges in the base visualization. (b) The *time lens* [Tom+12] shows the lens' selection and the generated lens result separately.

When the join is realized at early stages of the standard pipeline, the visual effect is often less confined. For example, as shown in Figure 6.12(a), the *Layout Lens* [TAS09] adjusts the positions of a subset of graph nodes to create a local neighborhood overview. Yet, relocating nodes implies that their incident edges take different routes, which in turn introduces some (limited) visual change outside the lens as well. Under a loose interpretation of the lens metaphor, the result of the lens function can even be shown separately. The *time lens* [Tom+12] depicted in Figure 6.12(b) is an example where the lens selection takes place in one part of the display (top-left circle around the mouse cursor) while the generated visual result is shown in another part (center).

A primary goal is to make it easy for the user to understand how the view seen through the lens relates to the base visualization. In addition to joining the lens effect with the base visualization, it is helpful to provide further visual feedback. For example, showing the lens border makes clear to the user that a lens is in operation and where the selection takes place. Dimming the base visualization is an additional visual cue that guides the user's attention to the effect shown within the lens.

An interesting question is how the join is carried out technically. A plausible solution is a three-step procedure. First, one renders the base visualization, optionally omitting the interior of the lens. Second, the result of the lens function is fetched to the lens interior, optionally blending with the base visualization. Third, additional visual feedback is rendered, including the lens geometry and optional user control elements. On modern GPUs, these operations can be accelerated via multiple render buffers, blending masks, and individual shader programs for the different steps.

However, this is not a universal solution. Instead the specific implementation details largely depend on the lens function and the desired effect. For example, the *Layout Lens* [TAS09] from Figure 6.12(a) is rather easy to implement as it only adjusts node positions at the stage of visual abstractions (VA). So the join $\bowtie$ is merely to override the default positions in the standard pipeline with the adjusted positions as computed by the lens. On the other hand, the join can be rather complicated, for example when considering different layers of 3D graphical information that need to be merged in correct depth order [KMS09].

In summary, selection σ, lens function, and join $\bowtie$ are the key components of the proposed conceptual model of lenses in visualization.

6.2.2.3. Discussion

In the previous paragraphs, key properties (shape, size, position and orientation) and key operations (selection σ, a lens function, and a join $\bowtie$) were identified which must be taken into account when designing and implementing an interactive lens in visualization.

It was indicated that a lens, described by its key properties, naturally partitions the display space into three regions: inside, outside, and border (see Figure 6.7). It was already mentioned that the lens effect is typically shown in the lens interior, whereas the base visualization in the lens exterior remains relatively unchanged. The border of a lens is equally important. Visually, the border clearly communicates the separation of lens interior and exterior, thus helping the user in defining the selection precisely. Later in Section 6.2.4 it is described that the lens border also plays an important role in terms of interaction. The lens border itself can serve as a handle to adjust the lens and it is a suitable place to attach additional user interface controls enabling the direct manipulation of lens parameters, as for example for the *FlowLens* [Gas+11].

A further aspect worth discussing is that the key operations relate to the stages of the data-state-reference model in different ways. The selection and join may gather input and feedback output at any stage. The lens function may generate the lens effect by implementing any subset of the data-state-reference model. There are two noteworthy observations. First, if the selection takes place at the early stages, the lens function has more options for creating a richer lens effect. For example, the *extended excentric labeling* [BRL09] selects data items which allows the lens function to generate textual labels and places them appropriately. The second observation is that the later the join with the base visualization is carried out, the more confined is typically the visual effect. Joins a the pixel stage as for the *Sampling Lens* [ED06b] are usually confined to within the lens, whereas joining at earlier stages as for the *Layout Lens* [TAS09] can have side effects on the whole base visualization.

Adding still more complexity, the proposed conceptual model in Figure 6.10 also indicates that it is possible to use multiple lens functions to generate a desired effect [Fox98]. The combination of lenses can be realized by iteratively attaching multiple lens pipelines to a standard pipeline or by combining lenses recursively. In the latter case, a lens pipeline is attached to another lens pipeline. However, combining lens functions requires proper execution of multiple interdependent selections and elaborate means for blending lens results in a conflict-free way [TFS08a]. All this is not easily accomplished and remains a challenge to be addressed in the future.

In the next section, the view on lenses is broadened from the geometric properties and conceptual model to actual lens techniques for different types of data and different tasks. Thereby the focus is set on lenses for graphs which support tasks dedicated to visual graph editing.

6.2.3. Categorization of Lenses according to Data and Tasks

Now, the focus is shifted to aspects that are more of interest to users of visualization tools by considering data and tasks. Such users may ask if there is a lens for the type of data and the tasks they have to accomplish. The following paragraphs aim to provide an answer to such questions. For the first part of the question, lenses for graphs and other types of data are investigated. For the second part of the question, lenses for different tasks are reviewed.

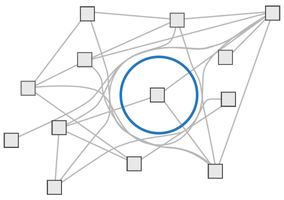
6.2.3.1. Lenses for Specific Data

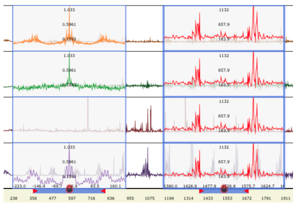
There are general purpose lenses and there are lenses that are specifically designed for a particular type of data. Lenses that operate on pixels or geometric primitives are usually oblivious to the specific data type of the underlying visualization. For example, magnification lenses (e.g., [Pin+12; FS05; CLP04; CM01; Kea98]) are techniques that are universally applicable across many types of data.

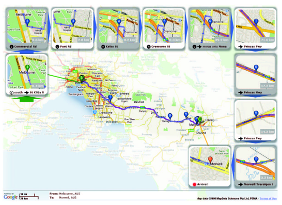
In this section, existing lens techniques which specifically address the characteristics of the visualized data are examined. Lens techniques addressing graphs are focused. The following survey of data-specific lenses is inspired by a slightly adapted version of SHNEIDERMAN'S taxonomy of data types [Shn96] that comprises:

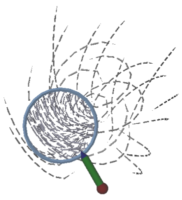
- **Graph data**
- Temporal data
- Geospatial data
- Flow data
- Volume data
- Multivariate data
- Text and document data

The purpose of the following paragraphs is to illustrate the diversity of existing lens techniques for graphs.

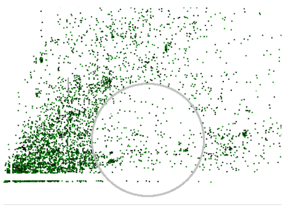

(a) Graph data

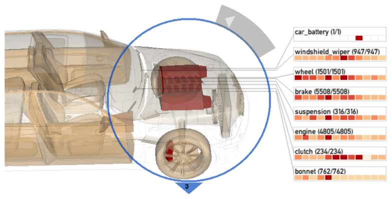

(b) Temporal data


(c) Geospatial data


(d) Flow data

(e) Volume data


(f) Multivariate data



(g) Text and document data

Figure 6.13.: Lenses for different data types. (a) *PushLens* [Sch+10] for graph data, (b) *ChronoLenses* [Zha+11] for temporal data, (c) *Detail Lenses* [Kar+10] for geospatial data, (d) magic lens for flow data [FG98], (e) *Magic Volume Lens* [Wan+05] for volume data, (f) *Sampling Lens* [ED06b; ED06a; EBD05] for multidimensional data, (g) lens widget for querying document data [CC13].

Graph Data Due to the fact that node-link representations are the most prominent visualization of graphs, it is not surprising that most lens techniques dedicated to graphs are designed for such representations. The effect of such lenses is often dedicated to *filtering out details* or to *enrich the representation with details*.

Although sophisticated graph layout algorithms exist, node-link representations might still suffer from edge clutter (i.e., a large number of edge overlaps and edge crossings), which hinders the analysis and editing of details. For example, if multiple edges fully overlap a certain node of interest, it cannot be recognized by the user anymore. Consequently, this node cannot be analyzed or edited too. Several lens techniques address the issue of edge clutter in node-link representations by filtering out irrelevant edges. The *EdgeLens* [WCG03] (and later the *PushLens* [Sch+10]) reduces the number of edges being visible inside the lens. The selection (σ) singles out edges that intersect with the lens geometry, but whose source and target nodes are not inside the lens. Then the lens interior is cleared of the selected edges by bending ($\bowtie$) them around the lens. Figure 6.13(a) illustrates edges that are routed around the *PushLens*. A similar effect is achieved by using the *Local Edge Lens* [Tom+06]. However, this lens filters out all edge segments crossing the lens which are not connected to a node inside the lens. Another lens following the idea of filtering out irrelevant edges is the *3DArcLens* [DSA15]. This lens works on node-link representations mapped to a virtual globe and totally clears the lens interior of any edge segments. While edges which cross the lens and are not connected to nodes inside are re-routed around the lens border, segments of edges connecting to nodes inside the lens are filtered out. Such lenses for filtering out details in node-link representations can be useful for graph editing in various ways. An example is the removal of edge clutter around a node of interest with the *Local Edge Lens* [Tom+06] for analyzing, and if necessary editing incident edges.

In contrast to graph representations with too many details, there also exist situations where a graph representation provides too few details for the task at hand. To support the user in this regard, different lens techniques for enriching the representation with details were developed. Van Ham and van Wijk propose a lens for smoothly expanding aggregated nodes to access more detailed levels of abstraction in a hierarchical graph [HW04]. A similar approach is presented in [Spi+10]. The *Network Lens* presented in [JDK10] can be used to enrich the representation of nodes with glyphs encoding additional node attribute values. Such lenses can also be useful in context of editing graphs. The *Network Lens* for example can be used to check nodes for missing attribute values which need to be corrected.

There are further lenses that address specific aspects of graphs. Dynamically transforming the global layout to generate local neighborhood overviews is the objective of the *Layout Lens* [TAS09] and the *Bring & Go* approach presented in [Mos+09]. Further examples include lenses specifically designed for displaying query results on hyper graphs [UČK13]. The *EditLens* introduced in Section 4.1 was particularly developed for editing the graph's structure.

Other Data Types For the sake of completeness, one exemplary lens for each of the other data types is described briefly. A more comprehensive overview of existing lenses for different types of data is provided in Table 6.1 on page 146.

Temporal Data: Data that have references to time are commonplace. Time series allow to understand the development of past events and predict the future outcome of ongoing phenomena. Analyzing time series often involves normalization, elimination of seasonal effects, and other temporal transformations. *ChronoLenses* [Zha+11] enable users to carry out such operations on-the-fly for selected time intervals (σ). The transformed part of the time series is superimposed ($\bowtie$) on top of the original version to maintain the context and facilitate comparison. As illustrated in Figure 6.13(b), *ChronoLenses* can be applied to multiple stacked time series simultaneously.

Geospatial Data: Data about the world often hold references to the geospatial domain, describing where certain phenomena or features have been observed. Maps are frequently used as a basis for visualizing the spatial frame of reference, which in turn serves to depict the actual geospatial data. As maps are omnipresent in everyday life, it is not surprising to find a lens developed for a routine problem: the depiction of driving directions. *Detail Lenses* [Kar+10] aim to make the visual representation of driving directions on maps easier to follow. To this end, relevant points of interest (POI) are defined (σ) along a route. For each POI, a lens with a detailed map of the POI is arranged around a global overview map ($\bowtie$) as illustrated in Figure 6.13(c). The arrangement is computed to ease association of lenses with the POIs along the route.

Flow Data: 2D or 3D vectors are the basic components of flow data. In addition to the raw vectors, derived scalar attributes, higher-level features (e.g., vortices), and flow topology are relevant aspects of the data. FUHRMANN and GRÖLLER [FG98] are among the first to propose lens techniques specifically for flow data. They use a planar 3D lens polygon (see Figure 6.13(d)) or a volumetric 3D lens box to implicitly or explicitly define a 3D spatial selection (σ) in the flow for which more details are shown. The low-resolution base visualization is replaced ($\bowtie$) by filling the lens interior with more and thinner streamlines that also have an increased number of control points.

Volume Data: Volume data are scalar data values on a three dimensional grid. Hence, lenses for volume visualization face the typical challenges related to spatial selection and occlusion in densely packed 3D visual representations. The *Magic Volume Lens* [Wan+05] addresses these challenges. In fact, the authors propose different lenses to magnify volumetric features, while compressing the context without clipping it entirely. Figure 6.13(e) shows a *Magic Volume Lens* applied to magnify a section of a human foot. The selection (σ) can be made interactively or can be adapted to data features automatically. The lens effect is directly embedded ($\bowtie$) into the direct volume rendering through a multi-pass rendering strategy.

Multivariate Data: General multidimensional, multivariate data typically span a frame of reference across multiple abstract dimensions. Along these dimensions multiple quantitative or qualitative variables are measured. For such data, dealing with data size is usually a challenging problem. The *Sampling Lens* [ED06b; ED06a; EBD05] addresses the problem of clutter in scatterplots and parallel coordinates. Through sampling, the lens function generates a more sparsely populated alternative view on the data. The selection (σ) made with the lens is inversely projected into the data space in order to estimate a suitable sampling rate for the alternative visual representation to be shown inside the lens. The clutter-reduced visual representation of the sampled data is rendered and clipped ($\bowtie$) at the lens boundaries as shown in Figure 6.13(f).

Text and Document Data: Text and document collections are rich sources of information. Analysis of text typically focuses on important topics or themes and querying relevant keywords. A particularly interesting application of a lens for exploring text databases is the approach by CHANG and COLLINS [CC13]. What makes their lens special is the fact that they combine spatial and abstract semantics in a single view. As illustrated in Figure 6.13(g), the lens selection (σ) operates on a 3D model of a car capturing the spatial semantics. Associated with the spatial selection are keywords which capture the abstract semantics. Keywords corresponding to the selection are visualized as heat map charts around the lens and an additional panel provides on-demand access to the underling text documents (⋈).

In sum, there are many different types of lenses for different types of data beside graphs. However, it would be interesting to investigate whether such lenses could also be useful in terms of supporting graph editing. For example, adapting the *Magic Volume Lens* to magnify features of a graph (e.g., nodes with missing or outlier attribute values) and to provide interactive means to edit these features (e.g., radial menus and radial sliders for editing node attribute values as presented in Section 5.2) could help in revealing and correcting erroneous data.

In the previous paragraphs, examples of lenses for different types of data were briefly reviewed. In the next section, lenses for specific tasks are examined.

6.2.3.2. Lenses for Specific Tasks

Lenses are designed as interactive tools. So it makes sense to review lens techniques in terms of the interaction tasks they do support specifically. In order to structure such a review, the seven key categories of user intents for interaction described by Yi et al. [Yi+07] are considered:

- Select
- Explore
- Reconfigure
- Encode
- Abstract/Elaborate
- Filter
- Connect

Each intent captures why a user wants to or needs to interact in the course of carrying out a certain task. When considering the tasks of visual graph editing described in Section 3.1, the intents appear as follows: The intent to select a certain data item is always present when carrying out the selection tasks of the category *Select or Specify Data to Edit*. However, the need for marking certain data items as interesting by selecting them can also arise when carrying out *Analyze Data* tasks. Often, such tasks require to view different parts of the graph. For that reason, the need for exploring the data can occur. The remaining intents (encode, abstract/elaborate, filter, connect, reconfigure) all have the overarching goal to interact with the system for adjusting the representation in a specific way. This includes obtaining certain details of interest. Thus, such intents are developed when accomplishing *Adjust Representation* tasks. A specific intent for interacting with the system in the course of carrying out *Apply Edit Operation* tasks is not included in the categorization from [Yi+07]. When performing such tasks, the user's intent is to interact in order to edit the underlying data. Thus, a further intent category is added:

- Edit

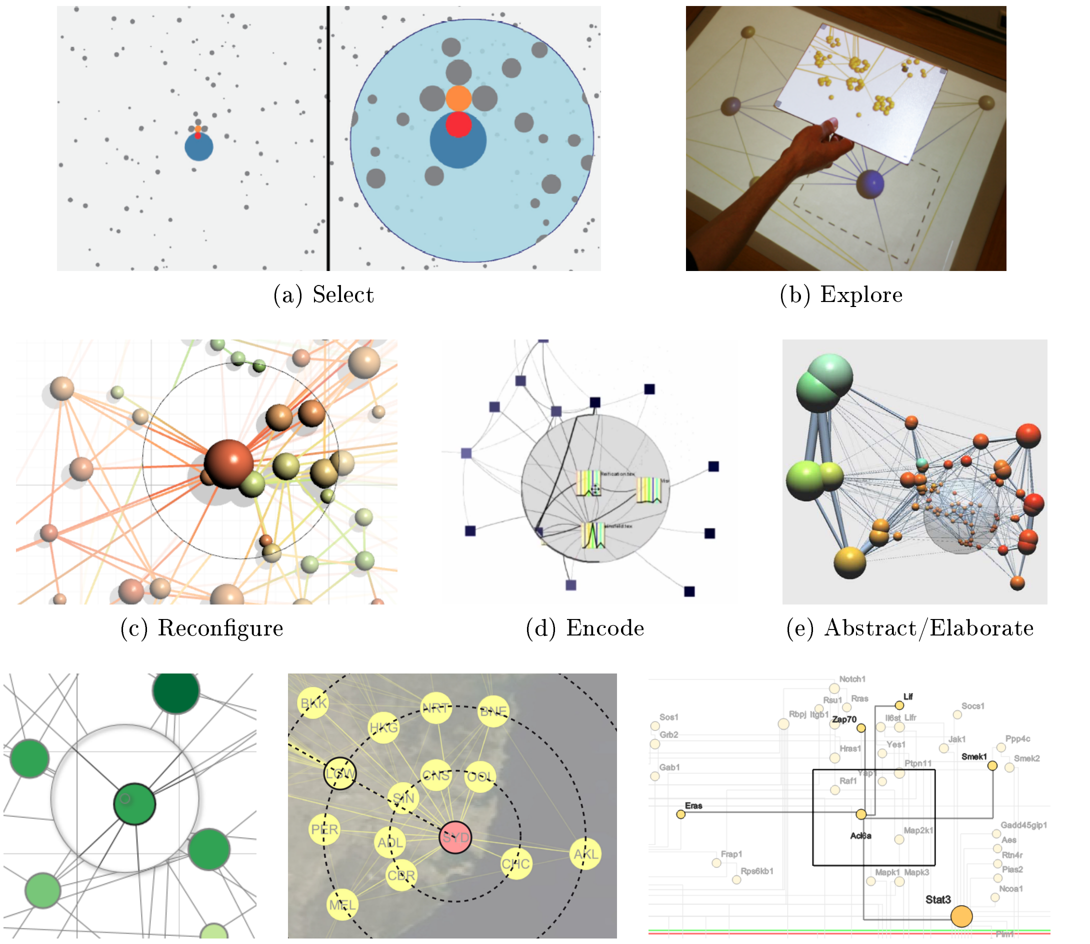
Similar to the review of lenses according to different types of data, exemplary lenses for each category are now very briefly described. This illustrates the utility of lenses for the different interaction tasks in context of visual graph editing. For a more complete list of lens techniques and their suitability for the different interaction tasks, the reader is referred to Table 6.1 on page 146.

Select – Mark Something as Interesting A typical problem in dense visual representations, is to pinpoint data items of interest. An example of a lens techniques supporting this task is the *Bubble Lens* [MW14]. This lens makes picking small data items easier, for instance nodes of a node-link representation, by actually enlarging not only the visual space in which visual feedback is shown, but also the motor space in which users interact. The magnification is automatically applied when the *Bubble Lens* is moved towards a region containing many small data items. Figure 6.14(a) illustrates the *Bubble Lens* entering such a region (left) and the subsequent magnified view (right).

Explore – Show Me Something Else Exploration relates to undirected search where the user is interested in seeking out something new. The *tangible views* for information visualization [Spi+10] demonstrate a variety of exploratory lens methods. As depicted in Figure 6.14(b), an example is the exploration of a graph at different levels of abstraction by sliding a tangible view through a virtual space-time above a tabletop display.

Reconfigure – Show Me a Different Arrangement The spatial arrangement of data items on the screen is generally the key to comprehending the visualized information. Looking at different arrangements helps to gain a better understanding of the data. A lens that temporarily rearranges the visual representation is the *Layout Lens* [TAS09] presented in Figure 6.14(c). The idea of this lens is to create a local overview of the neighborhood of the nodes inside the lens. For this purpose, the node layout is reconfigured locally to bring all relevant neighbors to a position within the lens interior.

Encode – Show Me a Different Representation The visual mapping decides about which parts of the data are visualized. However, it is often not possible to encode all aspects of the data within a single static image. By using the *Network Lens* [JDK10], the representation of nodes of a node-link visualization can be locally adjusted so that they additionally encode selected node attributes. Figure 6.14(d) shows an example where node attribute values are communicated through integrated parallel coordinate plots.



(a) Select (b) Explore

(c) Reconfigure (d) Encode (e) Abstract/Elaborate

(f) Filter (g) Connect (h) Edit

Figure 6.14.: Lenses for different tasks. (a) the *Bubble Lens* [MW14] assists in selection tasks, (b) *Tangible Views* [Spi+10] used to explore data, (c) the *Layout Lens* [TAS09] reconfigures nodes in a graph layout, (d) the *Network Lens* [EDF11] adapts the encoding of node attributes, (e) a lens for expanding and collapsing nodes of a graph [HW04], (f) the *Local Edge Lens* [Krü+13] filters out irrelevant edges, (g) *Bring & Go* [Mos+09] helps in connect tasks, (h) the *EditLens* edits the graph's structure (see Section 4.1).

Abstract/Elaborate – Show Me More or Less Detail This rather general task is supported in many different ways. Various general purpose lenses have been proposed that show more detail through magnification. The lens technique by VAN HAM and VAN WIJK [HW04] provides access to more or less details about clusters in a hierarchical graph. Nodes approached with the lens are automatically expanded, whereas nodes that are no longer in the focus of the lens are automatically collapsed.

Filter – Show Me Something Conditionally Filtering according to specific criteria, while still maintaining a global overview is an important task when exploring data. An example is the exploration of individual nodes and their connected edges in a node-link representation of a graph. In such situations, edge segments which are not connected to the nodes of interest become irrelevant and might even hinder analysis. By using the *Local Edge Lens* [Tom+06] as depicted in Figure 6.14(f), such edge segments can be filtered out.

Connect – Show Me Related Items Once the user has focused attention on data items of interest, the exploration usually continues with related or similar data. The *Bring & Go* approach [Mos+09] is a lens-like technique that supports navigation to related items in graphs. All nodes that are related (i.e., connected) to a focus node are brought close to the focus. Clicking on a connected node triggers an animation to that node. Figure 6.14(g) depicts an example where the airport Sydney (SYD) has been selected as the focus. Airports directly connected to SYD are arranged on concentric circles depending on their relative distance.

Edit – Change the data Interacting with the system for changing certain aspects of the data is essential in graph editing. The novel *EditLens* is a first lens which enables users to edit the structure of larger graphs directly in their associated node-link representations (see Section 4.1). Currently, there are no further lens techniques available supporting the user in this regard.

Already the limited number of approaches briefly mentioned in this section indicate that lenses are broadly applied in visualization for different data and different tasks. It is noteworthy, that a lot of lens techniques already exist which specifically address graph visualization. However, in terms of tasks of visual graph editing, the majority of these techniques only support the user in carrying out *Adjust Representation* tasks. Only few lenses facilitate tasks of the three remaining task categories. Hence, it remains to be investigated whether lenses can provide further support in this regard.

For details concerning the described lens techniques, the reader is referred to the original publications collected and categorized according to data type and task in Table 6.1.

This section presented lenses in terms of their semantics. The next section will focus on how users actually work with lenses. For this purpose, lenses are examined according to different interaction modalities and display settings.

	Data							Tasks							
Techniques	**Graph**	Temporal	Geospatial	Flow	Volume	Multivariate	Document	Select	Explore	Reconfigure	Encode	Abstract & Elaborate	Filter	Connect	Edit
[SB92; SB94] Fisheye Views	●		●									●			
[RM93] Document Lens							●					●			
[CMS94] MagicSphere					●			●			●	●	●		
[RC94] Table Lens						●						●			
[Vie+96] 3D Magic Lenses					●							●	●		
*[FG98] Lenses for Flow Visualization				●							●	●			
[FP99] Excentric Labeling						●			●			●		●	
[Sha+99] Interactive 3D Lens				●	●	●	●				●	●	●		
[LHJ01] Volume Magnification Lens					●							●			
[SFR01] Time Lens		●							●						
[Bak+02] Lenses for Genetic Networks	●									●	●	●			
[Mat+03] 3D Flow Lens				●							●	●			
*[WCG03] EdgeLens	●									●					
*[HW04] Graph Abstraction Lens	●											●			
[RHS05] Lenses in Geo-Environments			●								●		●		
*[EBD05; ED06b] Sampling Lens						●							●		
[RLE05] Temporal Magic Lens		●							●						
*[Wan+05] The Magic Volume Lens					●							●			
*[Tom+06] Local Edge Lens	●												●		
[KSW06] ClearView					●							●	●		
[Tra+08] 3D Generalization Lenses			●									●			
*[TAS09] Layout Lens	●									●				●	
[BRL09] Enhanced Excentric Labeling						●			●			●		●	
*[Mos+09] Bring &Go	●									●				●	
[ACP10] Precise Magnification Lenses	●	●	●	●	●	●	●	●				●			
*[JDK10] Network Lens	●										●				
*[Kar+10] Detail Lenses for Routes			●									●		●	
[Kin10] SignalLens		●										●			
*[Sch+10] PushLens	●									●					
*[Spi+10] Tangible Views	●		●			●		●	●	●	●	●	●	●	
[EDF11] Color Lens	●	●	●	●	●	●					●				
[Gas+11] FlowLens				●					●		●		●		
[Hei+11] SemLens						●					●		●		
[HTE11] MoleView	●		●			●				●					
[LWG11] Facet Lens							●				●		●		
[Pan+11] EdgeAnalyser	●					●					●	●			
[ZCB11] MagicAnalytics Lens		●												●	
*[Zha+11] ChronoLenses		●											●	●	
[Tom+12] Time Lens			●						●		●			●	
[Pin+12] JellyLens			●									●			
[Krü+13] TrajectoryLenses			●										●		
[Pin+13] Gimlenses					●							●	●		
*[UČK13] Lenses for Hypergraphs	●												●		
*[CC13] Lens for Querying							●						●	●	
[Alv+14] RouteLens			●									●			
[BHR14] PhysicLenses			●									●			
*[MW14] Bubble Lens						●		●				●			
[DMC15] VectorLens		●				●		●					●		
[KRD14] Multi-touch graph Lenses	●									●			●	●	
*[Gla+14] EditLens	●							●		●					●
*[DSA15] 3DArcLens	●		●							●			●		

Table 6.1.: Lens techniques in the context of visualization categorized according to data types and tasks. Entries are sorted in chronological order. Techniques for which a brief description is available in Section 6.2.3 are marked with *.

6.2.4. Lens Interaction and Display

An essential aspect of lenses, which enables a flexible use in the first place, is their interactivity. Interactive operations that have to be considered include creation and deletion of lenses, manipulation of the lens geometry (properties discussed in Section 6.2.2.1), as well as more complex operations, such as parameterizing the lens function or combining multiple lenses. The aim of this section is to briefly illustrate how lenses can be interacted with in terms of the following interaction modalities and display settings:

- Mouse+keyboard interaction in desktop environments
- Touch interaction on interactive surfaces
- Tangible interaction in physical space
- Body-controlled interaction in front of large displays

Mouse+Keyboard Interaction in Desktop Environments The most commonly used interaction modality and display setting for lenses is the traditional mouse+keyboard interaction on desktop computers. A major advantage of mouse input is the precision of the cursor movement. This is especially useful when precisely manipulating the lens geometry or when repositioning the lens. Such interactions are usually accomplished through drag and drop. Input through keyboard buttons can be used to trigger mode switches.

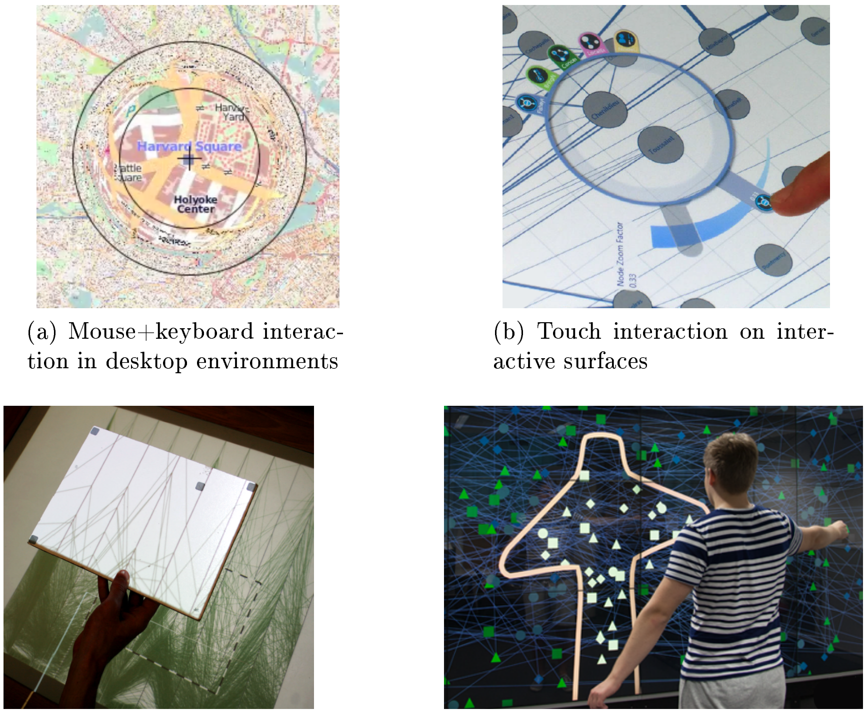
A common example of a lens used in desktop environments is the traditional magnification lens [Bie+93]. As exemplified in Figure 6.15(a), for such lenses, Appert et al. [ACP10] introduce mouse-interaction techniques to improve target acquisition.

Touch Interaction on Interactive Surfaces In the past decade, touch devices became increasingly commonplace. As these devices combine image and interaction space, they allow to interact with the visualized data in a more direct way. This advantage has been realized multiple times for the interaction with lenses.

Kister et al. for example introduce lenses specifically designed for touch-enabled devices [KRD14]. Their lenses can be repositioned through touch drag, scaled by applying a pinch gesture and rotated via two-finger rotation. For adjusting a lens' function and its parameters, Kister et al. propose externalized radial menus attached to the lens border (see Figure 6.15(b)) and continuous gestures directly applied to the lens.

Tangible Interaction in Physical Space Ishii and Ullmer [IU97] introduced the term tangible interaction as early as 1997. The idea was to use the affordances, tangibility, and manipulability of physical objects for an improved interaction with virtual data. This idea has been transmitted to lens interaction.

Spindler et al. present *tangible views* which are "[...] spatially aware lightweight displays that can be interacted with by moving them through the physical space on or above a tabletop display's surface" [Spi+10]. Tangible views have been used as physical



(a) Mouse+keyboard interaction in desktop environments

(b) Touch interaction on interactive surfaces

(c) Tangible interaction in physical space

(d) Body-controlled interaction in front of large displays

Figure 6.15.: Examples of interaction modalities and display settings used for lenses. (a) mouse interaction in a desktop setting to precisely position a magnification lens [ACP10], (b) touch interaction with a graph lens on a table top to adjust a lens function parameter [KRD14], (c) tangible interaction with a sampling lens in physical space [Spi+10], (d) body-controlled interaction with a graph lens in front of a large display [Kis+15].

lenses to support the exploration of different types of data, for instance multivariate data as illustrated in Figure 6.15(c). In this example, a sampling lens is used which can be repositioned through horizontal translation. The sampling factor of the lens function can be altered through vertical translation of the lens.

Body-controlled Interaction in Front of Large Displays On large high-resolution displays, controlling lenses with mouse and tangible interaction is infeasible. As presented in a recent work by Kister et al. [Kis+15], body-controlled interaction is a promising alternative. In their work, they introduce lenses specifically designed for wall sized displays and describe the rich design space of such lenses including their interaction. As an important part of the interaction, implicit horizontal movement of the lens position according to the user's position in front of the display is pointed out. Moreover, new concepts for defining the lens shape (e.g., using the user's virtual shadow as depicted in Figure 6.15(d)) as well as mappings of the user's distance to various lens parameters (e.g., lens size, vertical lens position, and lens function parameters) are introduced.

To conclude, different interaction modalities and display settings have been used to interact with lenses. However, the majority of existing lens techniques have been developed for mouse and keyboard interaction in desktop environments. As modern devices such as tablet PCs and large display walls become more and more commonplace, lens interaction dedicated to such devices requires further investigation. Moreover, as most lens interaction approaches focus on manipulation of the lens position and size, a possible direction for future work is the interactive adjustment of the lens shape, the lens function as well as lens function parameters.

In the next section, the examined aspects of lenses are discussed on the example of the *EditLens*.

6.2.5. A Demonstrating Example: the *EditLens*

In the previous sections, the focus was set on conceptual aspects of lenses as well as on providing examples for lenses concerning different data types, tasks, interaction modalities and display settings. Concrete lens techniques were covered only briefly, mainly for the purpose of illustrating the line of thought. This section proceeds in reverse. A concrete lens technique is focused and the aspects discussed in the earlier sections are summarized and illustrated by means of this technique.

Data and Task As described in Section 4.1, the *EditLens* is a lens technique which works in node-link representations of *graph data*. Concerning the overarching task it aims to support, this lens technique has an exceptional position. Although lens techniques usually support certain analysis tasks, the *EditLens* facilitates tasks dedicated to visual graph editing. Its mechanism for inserting and deleting nodes and edges enables users to *edit* the graph's structure. With the automatic suggestions of layout solutions for elements currently being inserted, it supports *Adjust Representation* tasks associated with the graph layout. This corresponds to the *Reconfigure* user intent category from Section 6.2.3.2. As the *EditLens* can also be used as a tool for either selecting elements to be edited or selecting regions in the graph layout, it also supports *Select or Specify Data to Edit* tasks, which is related to the user intent category *Select*.

The following discussion draws from the prototypical implementation of the *EditLens* for a touch-enabled desktop monitor as shown in Figure 6.16.

Properties and Model The *EditLens'* has a rectangular *shape* whose aspect ratio can be altered when needed. The lens' *size* and *position* can be set interactively too, whereas the lens *orientation* is predefined to match the orthogonal node-link layout. The *EditLens'* visual effect is to embed layout suggestions for nodes and edges to be inserted based on the local region defined by the lens properties as well as the current graph layout. To

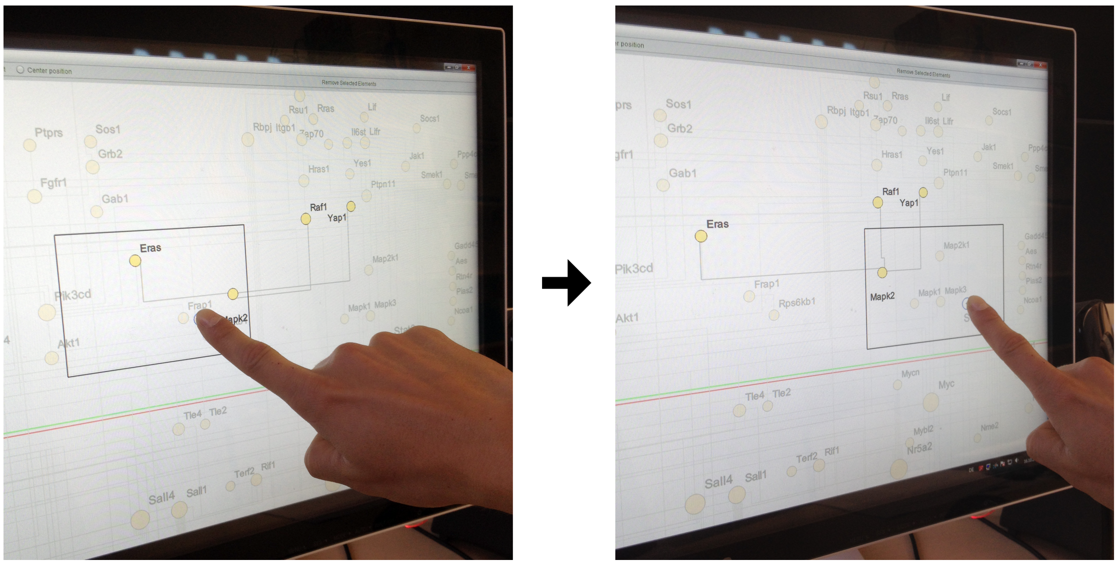


Figure 6.16.: Repositioning the *EditLens* through direct touch interaction on a multi-touch enabled desktop display.

achieve this effect node positions and edge routes must be computed. Thus, in terms of the conceptual model of lenses, the *selection* (σ) must pick the data to be inserted at the stage of *visual abstraction* at the latest. The *lens function* thereafter computes an optimized layout solution for the selected data with respect to existing layout constraints. This computation considers the currently chosen node placement strategy as a parameter for the lens function. Afterwards, the lens' *join* ($\bowtie$) can be implemented at the stage of *visual abstraction* so that the node-link representation of the graph, *enriched* with the computed layout solution of the selected data, can be rendered afterwards. Alternatively, the join could also be realized on the stage of *image data*. This requires blending the result of the lens function with the result of the standard node-link visualization.

The *EditLens'* effect on the data, namely the insertion or deletion of nodes and edges is triggered through user interaction.

Display and Interaction As indicated in Section 6.2.4, the display environment and the interaction modality are key factors influencing the way users work with lenses in visualization applications. The *EditLens* prototype is specifically designed for a single multi-touch display setup fitting for a desktop scenario. The display used for evaluating the *EditLens* could be tilted to obtain a comfortable working position. The visualization (including the lens) is shown in full-screen and can be interacted with either traditional and familiar mouse+keyboard interaction or new and more direct multi-touch gestures. This includes creating, removing, repositioning (see Figure 6.16) and resizing the lens as well as inserting or removing nodes and edges with the lens. For concrete gestures and their effect, the reader is referred to Section 4.1.2.5.

In summary, the described example of the *EditLens* is the first lens technique for editing graph data through their visual representation. Further lenses for editing data, for instance the graph's attribute values as indicated in Section 6.2.3.1 are possible. Due to their focused, lightweight, and temporary adjustments of an existing visualization, a lens acts as an individual tool which moreover can co-exist with several other lenses in the same visualization.

6.2.6. Discussion

In this Section, lenses were investigated to better support the user in analyzing or editing certain details of interest. As a large variety of lens techniques already exists, a comprehensive overview of lenses was provided. For that, lenses were defined first and their geometric properties were examined. To better capture how lenses actually work, a conceptual model of lenses was introduced according to which lenses are defined as additional lens pipelines attached to the standard visualization pipeline of the data-state-reference model. In this model, a visualization lens can take effect at any stage of the transformation from the raw data values to their visual representation. The wide applicability of lenses in visualization contexts was illustrated by a systematic review of lens techniques for different types of data and different user tasks. Thereby the focus was set on graph data and tasks related to visual graph editing. As the flexible use of lenses is mainly attributed to interaction, different possibilities of lens interaction in terms of interaction modality and display setting were surveyed. At the end of this section, all examined aspects of lenses were discussed concerning an exemplary lens technique, the *EditLens*.

Throughout this overview, possibilities for future work concerning lenses for visual graph editing could be identified. Although several lens techniques working on graph data already exist, most of them specifically support the user in carrying out *Adjust Representation* tasks. Only the new *EditLens* developed within this thesis further addresses *Apply Edit Operation* tasks. Hence, it remains to be investigated whether lenses can be further utilized in this regard.

Moreover, it is noticeable that virtually all lenses specifically designed for graph visualization work on node-link representations. However, when editing graphs via matrix representations, such techniques would be helpful too. Thus, another direction for future work is to adapt existing lens techniques for matrix representations or to develop new solutions in this respect.

The following options for future research address lenses in general. While existing lens techniques could be categorized, these are only first attempts at structuring the design space of lenses. In the future, this overview could be used as a basis to develop a formal in-depth taxonomy of lenses in visualization, and maybe of lens techniques in general.

Virtually all reviewed lenses support direct manipulation of the lens position on the screen. However, adjusting other properties such as size, shape, or parameters controlling the lens function often need to be carried out by external controls. Moreover, the flexible combination of lenses to create lens functions on-the-fly has not been sufficiently addressed yet. Both aspects require more studies on the interaction side of lenses. As indicated in the previous section, utilizing modern interaction modalities seems to be a promising direction in this regard.

6.3. Summary

In this chapter, user support with regard to the following two issues was considered: First, due to the limited screen size and resolution, users have to navigate in order to view and edit different parts of the graph representation. This however can be difficult due to the large number of navigation possibilities and the unawareness of the user where interesting data for the task at hand could be located. Second, due to the size and complexity of graph data, users have to either deal with representations visualizing a subset of the data only or representations which are cluttered due to too many information. This can hinder the analysis or editing of details.

To facilitate the user's navigation decisions and to mitigate the trial-and-error character of navigation, a novel approach for providing navigation recommendations to interesting targets of a hierarchical graph was introduced. This approach includes a flexible definition of interestingness based on the notion of a degree of interest which allows recommending horizontal navigation in terms of the graph layout and also vertical navigation in terms of the level of abstraction. The actual recommendations are communicated through visual cues embedded into the visual graph representation. It has been demonstrated how the approach can be generalized to the concept of event-based visualization [Tom06]. Although a proof-of-concept implementation indicates the usefulness of this approach, a formal evaluation is left for future work. As described in the use case in Section 6.1.4, the proposed approach assists the user in exploring data to be edited.

To better support the user in situations where details must be analyzed or edited, the concept of lenses was investigated. Within a comprehensive overview of existing lens techniques, lenses for different data types and different tasks were surveyed. Thereby, the focus was set on lenses for filtering and providing details of graph data and lenses for supporting tasks related to visual graph editing. Additionally it has been indicated that existing lens techniques for other data types and tasks can be adapted to facilitate the editing of graphs.

Moreover, different modalities and display settings used to interact with lenses were examined. To better understand the concept behind lenses, key properties of lenses were described and a general model of lenses was established, which describes how lenses are integrated into the visualization process. All the considered aspects of lenses were summarized and illustrated by means of the *EditLens* developed within this thesis.

After having investigated tasks of visual graph editing, techniques for editing the graph's structure and the graph's attributes as well as user support, the following chapter summarizes this thesis and details unsolved questions to be addressed in future work.

7. Summary and Outlook

Summary

Visualization approaches are traditionally used for analyzing graphs. This thesis investigates how to use these approaches to additionally support the editing of graphs. The motivation behind this idea is simple. Visual representations of the data support the user concerning two important aspects: They facilitate the discovery of data items relevant for editing, and they can communicate the changes to the data caused by edit operations. Hence, it is beneficial for the user if the editing is carried out directly in the visual representation. However, as the literature review concerning graph visualization and graph editing in Chapter 2 shows, only few approaches for editing the structure of small graphs in node-link representations exist. For the structural editing of larger graphs and the editing of the graph's associated attribute values in general, visualization approaches have not been considered so far. Moreover, the tasks a user has to carry out when editing graphs visually have not been investigated comprehensively before. Thus, the goal of this thesis was to provide a conceptual view on tasks of visual graph editing and to develop novel techniques based on graph visualization for their interactive support.

In Chapter 3, tasks of visual graph editing were described and categorized, and their current support by existing visualization and interaction techniques was reviewed. It was determined that especially tasks for applying edit operations are currently poorly supported. To describe the interplay of all tasks involved in visual graph editing, the *general editing model* from [Bau06] was extended to further include the tasks for analyzing the data and specifying new data to be inserted. The resulting *extended general editing model* allows describing a large variety of visual graph editing workflows.

Based on this comprehensive overview of tasks of visual graph editing, new approaches supporting tasks for editing the graph's structure (Chapter 4) and the graph's attribute values (Chapter 5) were developed. Although various tasks were addressed within these approaches, a special focus was set on applying edit operations. Exemplary editing workflows were described in the light of the *extended general editing model.* To cover the different classes of graph representations reviewed in Chapter 2, these novel approaches are based on node-link and matrix representations with attribute-independent and attribute-dependent layout.

The first approach presented in Chapter 4 specifically addresses the structural editing of larger graphs with customized and established node-link layouts. The key idea is to use an interactive *EditLens* which defines a region where an edit operation affects the graph layout and the underlying structure. Locally optimal node positions within the lens and edge routes to connected nodes are calculated automatically according to different criteria. In contrast to layouting nodes and edges by hand, this spares the user much manual work, but still provides sufficient freedom to accommodate application-dependent layout constraints.

The second approach introduced in Chapter 4 addresses the structural editing of graphs in matrix representations, which have not been considered for graph editing before. First interaction techniques for inserting and deleting single as well as multiple edges are presented. Moreover, an approach for using node-link and matrix representations in combination to exploit their individual benefits is introduced.

In the first part of Chapter 5, a novel approach supporting the editing of node attribute values in node-link representations with scatterplot and semantic substrates layout is presented. Both layouts are attribute-dependent. To support the user in determining nodes relevant for editing, nodes with missing values and outliers are visually emphasized. Attribute values of one or multiple nodes can be edited directly by changing the nodes' position in the graph layout via drag and drop gestures. To allow for sufficient interaction precision and to reduce interaction effort, scatterplot layouts were extended with a focus+context magnification technique and semantic substrates layouts were extended with an off-screen visualization. Moreover, visual feedback for communicating changes to the locally edited data value and the global value distribution was integrated. This allows the user to observe the effect of applied edit operations.

In the second part of Chapter 5, a novel attribute editing approach based on node-link representations with attribute-independent layout is presented. The graph's structure is represented with the node-link layout and associated attribute data is communicated by varying the appearance of nodes and edges in color and shape. Missing values and outliers are represented in sharp contrast to other values for supporting their identification. For editing node or edge attribute values, a pop-up editing interface can be invoked on the element of interest. This interface is integrated into the representation and provides controls for editing qualitative and quantitative attribute values. To make the user aware of the local editing effect, the visual appearance of the edited element is constantly updated. Effects of edit operations to the global value distribution are communicated with visual cues attached to the editing interface. With the approaches presented 5, the visual editing of graph attribute values becomes possible.

As the introduced approaches to edit graphs are based on their visual representations, there might be situations were the editing or analysis of certain details requires navigation or an adjustment of the representation. To better facilitate the user's navigation decisions and to reduce the trial-and-error character of navigation, an approach for providing navigation recommendations to interesting targets is presented in the first part of Chapter 6. As this approach uses a generalization of the user's current focus, navigation recommendations to a broad range of interesting targets become possible. In contrast to existing solutions, this includes navigation recommendations to interesting nodes which are disconnected from the currently focused set of nodes. Another novelty of this approach is the consideration of both, horizontal navigation recommendations in the graph layout and vertical navigation recommendations in terms of the level of abstraction in a hierarchical graph.

To support the user in adjusting the representation in terms of filtering out or providing additional details, the concept of lenses was investigated in the second part of Chapter 6. Within a comprehensive overview of existing solutions, lenses for filtering or providing additional details of graphs as well as lenses which specifically support certain tasks related to visual graph editing were surveyed. Moreover, it has been indicated that lenses for other data types and tasks can be adapted to facilitate visual graph editing. To provide a conceptual view on lenses, key properties of lenses were described and a general model of lenses was established. This facilitates the development of novel lens techniques for graph editing like the *EditLens*.

In summary, this thesis introduces a unified view on graph analysis and graph editing. To support the users in carrying out both tasks seamlessly, various approaches for integrating editing facilities into visual graph representations commonly used to analyze the data were developed. By using these approaches, cognitively expensive switches between different working environments for carrying out either analysis or editing tasks are not necessary anymore. This is beneficial for the user in various situations. If the graph visualization reveals that the dataset must be edited (e.g., for correcting it, or for cleansing it in order to enable a meaningful analysis), data changes can now be carried out directly. Furthermore, as the visual representation communicates data changes immediately and in an effective way, the user is supported in analyzing the effect of applied edit operations. This for example allows testing different what-if-scenarios.

Outlook

Although visual graph editing was comprehensively discussed within this thesis, there is room for future work with regard to four major issues:

Base Visualization The newly introduced graph editing approaches are based on different visual graph representations. However, besides these representations, existing alternatives remain to be investigated in terms of supporting graph editing. In this regard, representations which have become established in certain application domains or for visualizing certain types of graphs are of particular interest. An example are implicit representations for trees.

The proposed approaches for editing the graph's attribute values provide visual cues for communicating values potentially relevant for editing, as well as the effect of edit operations to the global data context. Depending on the application, such information can also be valuable when editing the graph's structure. For example, when editing graphs based on traffic networks, disconnected nodes or isolated subgraphs, which cannot be reached from other nodes, might be potential candidates for editing and could be emphasized visually with a dedicated graph layout. Moreover, information about how important, global graph properties changed after applying certain edit operations (e.g., the graph's connectedness or average shortest path length) could be communicated with additional visual cues. The development of suitable solutions in this respect should be addressed in future work.

Besides communicating the consequences of individual edit operations, another interesting aspect worth investigating is the communication of the effect of a series of edit operations, e.g., the editing history. With a dedicated visualization approach, the user could be supported in understanding how the data developed over time. Such insight might influence further editing decisions.

Graph Data In this thesis, the editing of single nodes, edges and attribute values was focused. Although first approaches for editing sets of edges and attribute values were introduced too, appropriate support for edit operations on sets requires further in-depth investigation. Especially challenging in this regard is to provide support for flexibly applying update operations on sets of attribute values. This requires providing interactive means for defining how values are being changed.

Another aspect worth investigating in future work is the simultaneous editing of multiple graphs. Various application areas exist where users have to work with sets of graphs at the same time. Examples include individual time steps of dynamic graphs or multiple graphs resulting from simulation runs with different parameters (e.g., when using climate models). When such sets of graphs must be edited in a consistent way, it is cumbersome and error-prone to apply the same edit operations to every individual graph. Thus, solutions for visually editing sets of graphs must be developed which include an automatic propagation mechanism for edit operations.

Interaction As described in Chapter 3, the visual editing as well as analysis of graphs includes various tasks which have to be carried out interactively. In traditional desktop settings, usually mouse and keyboard input are used for this purpose. However, when working with modern devices like tablets, tabletops or display walls, which have become increasingly important in visualization research, mouse and keyboard interactions become ineffective. In such situations, interactions suitable for the display setting must be utilized. Besides touch interaction, which has been used for most of the solutions presented in this thesis, pen interaction, gaze interaction, body-centric interaction and combinations thereof should be investigated for graph editing in future work. However, designing a suitable set of interactions is a challenging task as presented in [Gla+15b; Gla+15c].

Evaluation Although qualitative user feedback concerning some of the proposed approaches was already collected within user studies, not all approaches have been evaluated this way. Consequently, a necessary step for future work is to carry out further user studies. Besides collecting qualitative feedback, a comparison between the proposed approaches and standard alphanumeric editing technique, with regard to quantitative measures like editing speed and error rates, would be interesting. As such user studies are difficult to design, collaboration with evaluation experts is suggested.

The applicability of every approach introduced in this thesis was illustrated by a working implementation applied to a use case of a certain application domain. It remains to be tested, whether the proposed approaches are also useful for visually editing graph data from different application domains.

8. Appendix: An Orthogonal Edge Routing Algorithm for the *EditLens*

The *EditLens* introduced in Section 4.1 requires an edge routing algorithm for a node-link representation that is able to compute orthogonal edge routes upon incremental changes to the graph in interactive frame rates. Since existing algorithms do not meet these requirements a novel orthogonal edge routing algorithm on the basis of the algorithm from [WMS09] was developed. For this algorithm it is assumed that nodes are represented with overlap-free circles whose quadratic bounding boxes leave a certain distance between each other. The algorithm consists of the following three steps which are described in detail below.

1. Computation of potential edge routes
2. Search of suitable edge routes
3. Resolution of edge overlaps

For simplification, the terms node and edge refer to their visual representation in a node-link layout in the following, unless otherwise stated.

Computation of Potential Edge Routes

The first step of the algorithm is to determine all potential orthogonal edge routes of the node-link representation. For that, an auxiliary structure – the so-called *orthogonal visibility graph* (OVG) – is computed. This structure describes all potential orthogonal edge routes and thus allows finding certain routes by applying search algorithms. As this structure is based on the nodes, it must be updated upon every insert, delete or refresh operation on nodes.

Every node has a primary connector point in its center to which all incident edges connect and four secondary connector points on each corner of its bounding box, through which an edge can be routed. The connector points are the nodes of the OVG. The orthogonal edges of the OVG are calculated incrementally for pairs of nodes of the OVG. Let $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ be two nodes of the OVG whereas v_1 belongs to the node n_1 of the node-link representation and v_2 to the node n_2. Now the two orthogonal lines $e_1 = (v_1, (x_1, y_2), v_2)$ and $e_2 = (v_1, (x_2, y_1), v_2)$ are computed. If $x_1 = x_2$ or $y_1 = y_2$, than e_1 and e_2 are reduced to one edge $e_3 = (v_1, v_2)$. For e_1, e_2, or e_3 it is tested, whether they intersect with bounding boxes of existing nodes except n_1 and n_2. Non-intersecting

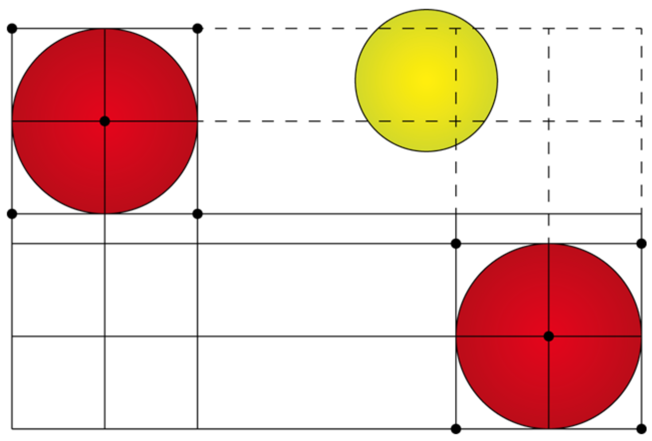


Figure 8.1.: Intersection test of orthogonal lines between the connector points of the red nodes. Only the solid lines become edges of the OVG as the dashed ones intersect with the bounding box of the yellow node.

lines become edges of the OVG. Figure 8.1 illustrates the computation of orthogonal lines for the two red nodes. In this case, only the solid lines become edges of the OVG. The dashed lines cannot be used because they intersect with the bounding box of an existing node (depicted in yellow).

In contrast to the OVG of [WMS09], which only contains segments of potential orthogonal edge routes, the edges of the OVG used here can be clearly assigned to two nodes of the node-link representation (connector points have a reference to their nodes). This property is important for efficiently updating the OVG upon insert, delete and refresh operations. If nodes are inserted, the dedicated nodes and edges of the OVG can be inserted easily as described above. If nodes are deleted or refreshed, all affected edges of the OVG can be deleted in $O(|E_{OVG}|)$, whereas E_{OVG} is the set of edges of the OVG. If nodes are placed on existing edges of the OVG upon insert or refresh operations with the *EditLens* (verified with an intersection test), the respective edges of the OVG are marked as blocked and excluded from the following search. An example is depicted in Figure 8.1. Assuming that the yellow node is placed after the red ones, the dashed edges are marked as blocked. In case the blocking node is placed on a new position upon refresh, it is tested whether the blocked edges of the OVG are still blocked and released if necessary. If the blocking node is deleted, the blocked edges are released too.

The OVG used here and the OVG used by Wybrow et al. describe identical potential edge routes. However, the OVG used here requires more memory space. The next step is to use the OVG to search for suitable edge routes.

Search of Suitable Edge Routes

With the OVG, the search for a suitable edge route between the primary connector points of two arbitrary nodes is easy. If multiple edge routes must be found, the search must be performed multiple times. For this purpose, a search algorithm for graphs can be used. It is suggested to prefer fast, informed search algorithms like *A-Star*[1] [HNR68], which is also utilized in WYBROW ET AL.'s algorithm. For finding edge routes that are short and have a low number of edge bends, the cost function of *A-Star* must be set up accordingly. In case of large OVGs the search should be restricted temporally. If no route can be found in a certain time (e.g., no route exists and the whole OVG must be traversed), a fall back solution must be used or the user must be prompted for further interaction. After finding an edge route, its final representation must be computed.

Resolution of Edge Overlaps

The search for suitable edge routes on the OVG provides edge routes that might overlap partially. Using these routes in the node-link representation would introduce ambiguities and hinder tracing edges. For that reason a method called *nudging* is used to resolve these overlaps. The nudging used in [WMS09] uses available space to spread edges as much as possible and a global strategy to minimize their edge crossings. With this nudging, good results can be achieved. However, it's runtime is too long for interactive frame rates. Thus, a simple local nudging is used here. All edge routes provided by the search are shifted with an offset in x and y direction. For avoiding crossings with existing nodes due to this shift, the bounding boxes of every node must be enlarged by a safety gap δ. As this is already done when inserting nodes with the *Editlens*, no further adjustments are needed. As illustrated in Figure 8.2, this introduces lanes between all nodes with a thickness of a least 2δ. Within these lanes, the final edge routes are laid out. The offset of the shift is computed as follows: Let r_{src} and and r_{dest} be the radii of the source and the target node of an edge to be routed. Let *rand* be a random number generator for the interval defined by its arguments and let *min* be the minimum function.

$$
\begin{aligned}
offset_x &= rand(0, min(r_{src}, r_{dest}, \delta - \mu)) \\
offset_y &= rand(0, min(r_{src}, r_{dest}, \delta - \mu))
\end{aligned}
$$

μ is a parameter > 0 that prevents edge routes from touching the border of nodes except for the source and target node. By shifting the edge routes with this offset, overlaps are often resolved and it is assured that the routes will connect to the circular nodes. Figure 8.3(a) presents a good result of this method during the insertion of a node with incident edges using the *EditLens*. However, due to the randomness of the shift, overlaps can still appear in some cases. An example where this might happen are nodes

[1] *A-Star* is an informed search algorithm for graphs which is utilized to find the least-cost path between two given nodes of a graph. To enable a targeted and fast search, a heuristic cost function is used.

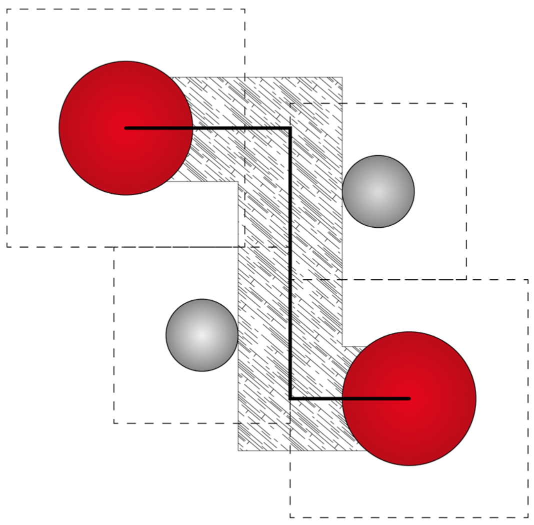


Figure 8.2.: Lanes between nodes used for edge routes (depicted as hatched area) resulting from enlarging the nodes bounding boxes by a safety gap δ.

with many incident edges. Additionally, edge crossing introduced by this shift are not resolved. This can be seen in Figure 8.3(b). Nevertheless, the user is free to explore different results with the *EditLens* until a good one is found where these problems do not (or only marginally) occur.

In [Roh+10], another local nudging strategy based on a topological ordering of a directed acyclic graph is used. It needs to be tested, whether this strategy is a suitable alternative.

Discussion

The proposed edge routing algorithm has been implemented in the *EditLens* prototype. In order to achieve short runtimes, efficient data structures and parallel processing must be utilized.

For managing nodes, *R-Trees* are suitable. This hierarchical data structure offers an efficient intersection test needed for computing and updating the OVG. This intersection test can be realized for each segment of an edge route. Another positive aspect of *R-trees* is that they provide important methods for inserting and deleting nodes, which are needed for incrementally constructing the graph layout.

The storage and the management of the OVG are difficult to handle. On the one hand, inserting and deleting its nodes and edges must be possible efficiently without recomputing the whole structure from scratch. For this purpose, linked lists can be used for example. On the other hand, the OVG should fit into the RAM to prevent slow access times due to outsourcing to the hard disk. This is already challenging with graphs of the *PluriNetwork's* size, as the number of edges of the OVG rises quadratically with its number of nodes. For this reason, only necessary information should be stored in the OVG.

For searching individual edge routes in the OVG, individual threads can be used. This way, the search can be parallelized, which effectively reduces its runtime.

To verify that the proposed edge routing algorithm is significantly faster than the algorithm from WYBROW ET AL., it was tested under the same conditions (see Section 4.1.2.4). The average results of multiple runs are illustrated in Figure 8.4(a). It shows, that even with increasing number of nodes, the time needed to compute the two orthogonal edge routes remains constant at about 0,4 ms. For comparison, the test results of the algorithm from WYBROW ET AL. are depicted in Figure 8.4(b). This figure shows that the runtime increases rapidly with increasing number of nodes. The computation of two edge routes for a node-link representation with 65 nodes and 129 edges already took 100 ms.

Due to the short runtime of the new edge routing algorithm, the insertion of nodes with 15 or even more incident edges (as depicted in Figure 8.3(b)) is often still possible in interactive frame rates when utilizing the *EditLens*. However, there are also exceptional situations where the algorithms cannot compute an edge route in a given time limit (e.g., 500 ms). The reasons for that are either non-existing edge routes or an extraordinary amount of nodes that must be circumvented. With increasing number of incident edges upon node insertion, the node placement strategies can also be the limiting factor (e.g., *Edge Length First*, see Section 4.1.2.1).

After developing this algorithm, MARRIOT ET AL. presented a novel orthogonal edge routing algorithm which is also significantly faster than the algorithm from WYBROW ET AL [MSW14]. It remains to be tested whether this algorithm is an alternative to the proposed algorithm.

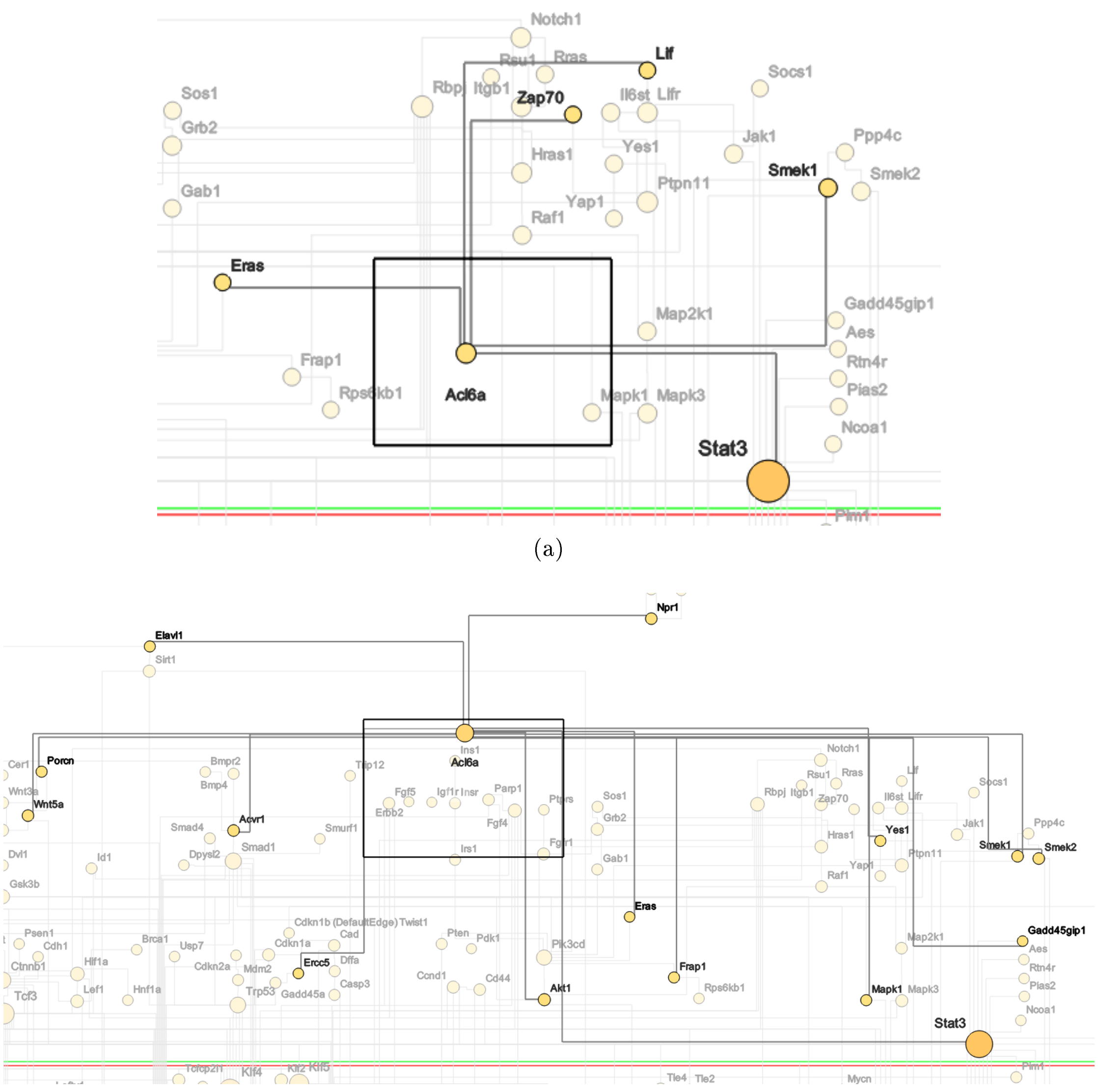
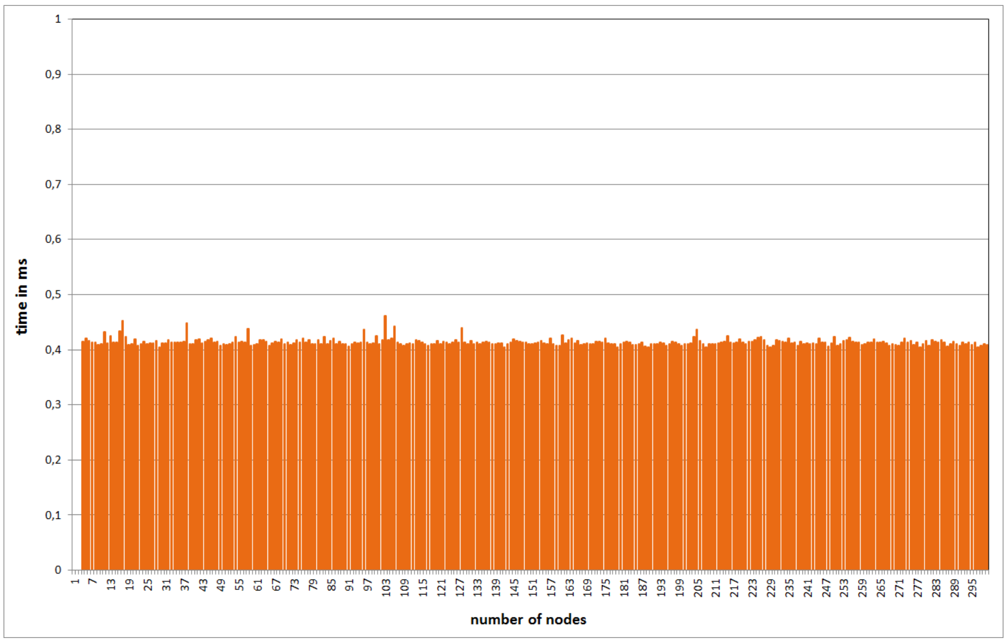
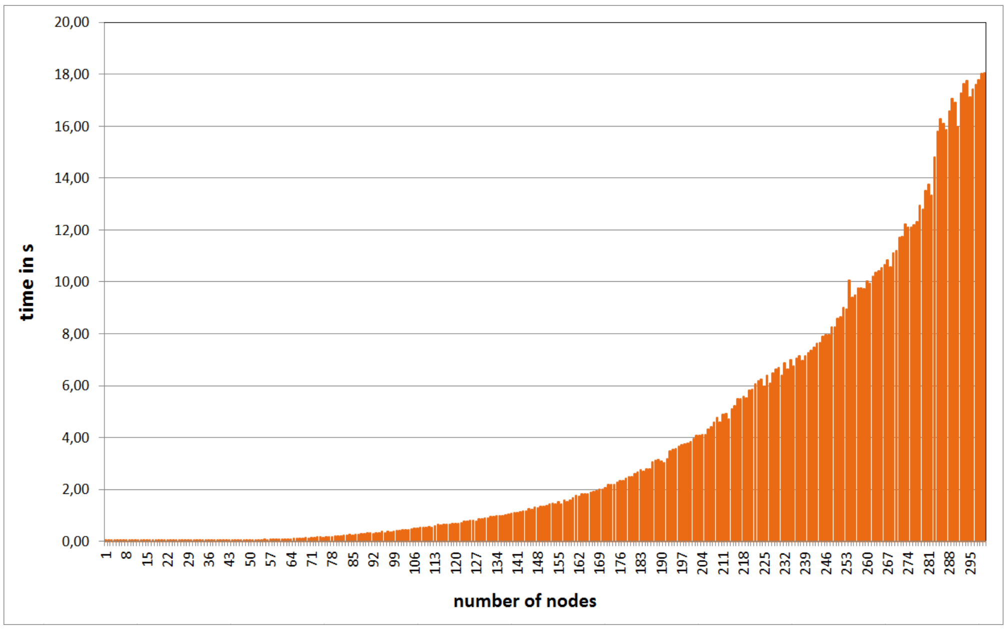


Figure 8.3.: Two results achieved by the proposed edge routing algorithm. Routing suggestions of 5 edges (a) and routing suggestions of 15 edges (b) during insertion of a node with the *EditLens*. The green and red lines depict borders of semantic regions of the *PluriNetwork* (see Section 4.1.4.1).



(a)



(b)

Figure 8.4.: Results of a runtime performance test for computing orthogonal edge routes of a node-link layout with (a) the edge routing algorithm proposed in this thesis and (b) the edge routing algorithm from [WMS09].

Bibliography

[Abe+14] J. Abello, S. Hadlak, H. Schumann, and H.-J. Schulz. "A Modular Degree-of-Interest Specification for the Visual Analysis of Large Dynamic Networks". In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2014), pp. 337–350.

[ACP10] C. Appert, O. Chapuis, and E. Pietriga. "High-Precision Magnification lenses". In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2010, pp. 273–282.

[AES05] R. Amar, J. Eagan, and J. Stasko. "Low-Level Components of Analytic Activity in Information Visualization". In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* Washington, DC, USA: IEEE Computer Society, 2005, pp. 111–117.

[AH04] J. Abello and F. van Ham. "Matrix Zoom: A Visual Interface to Semi-External Graphs". In: *Proc. of International Conference Information Visualisation (IV).* 2004, pp. 183–190.

[AK07] K. Andrews and J. Kasanicka. "A Comparative Study of Four Hierarchy Browsers using the Hierarchical Visualisation Testing Environment (HVTE)". In: *Proc. of International Conference Information Visualisation (IV).* 2007, pp. 81–86.

[Alv+14] J. Alvina, C. Appert, O. Chapuis, and E. Pietriga. "RouteLens: Easy Route Following for Map Applications". In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI).* ACM, 2014, pp. 125–128.

[AMA07] D. Archambault, T. Munzner, and D. Auber. "TopoLayout: Multi-level Graph Layout by Topological Features". In: *IEEE Transactions on Visualization and Computer Graphics* 13 (2007), pp. 305–317.

[AN05] J. Arvo and K. Novins. "Appearance-preserving Manipulation of Hand-drawn Graphs". In: *Proc. of International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia.* ACM, 2005, pp. 61–68.

[AN06] J. Arvo and K. Novins. "Fluid Sketching of Directed Graphs". In: *Proc. of Australian User Interface Conference.* Australian Computer Society, Inc., 2006, pp. 81–86.

[And+10] G. Andrienko, N. Andrienko, U. Demsar, D. Dransch, J. Dykes, S. I. Fabrikant, M. Jern, M.-J. Kraak, H. Schumann, and C. Tominski. “Space, Time and Visual Analytics”. In: *International Journal of Geographical Information Science* 24.10 (2010), pp. 1577–1600.

[Aub04] D. Auber. “Tulip : A huge graph visualisation framework”. In: *Graph Drawing Software*. Ed. by P. Mutzel and M. Junger. Springer, 2004, pp. 105–126.

[BA11] D. Bonnet and C. Appert. “SAM: The Swiss Army Menu”. In: *French Speaking Conference on Human-Computer Interaction*. ACM, 2011, pp. 51–54.

[Bak+02] C. A. H. Baker, M. S. T. Carpendale, P. Prusinkiewicz, and M. G. Surette. “GeneVis: Visualization Tools for Genetic Regulatory Network Dynamics”. In: *Proc. of IEEE Visualization Conference (Vis)*. IEEE Computer Society, 2002, pp. 243–250.

[Bat+98] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Inc., 1998.

[Bau06] T. Baudel. “From Information Visualization to Direct Manipulation: Extending a Generic Visualization Framework for the Interactive Editing of Large Datasets”. In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 2006, pp. 67–76.

[BCS04] T. Bladh, D. Carr, and J. Scholl. “Extending Tree-Maps to Three Dimensions: A Comparative Study”. In: *Computer Human Interaction*. Ed. by M. Masoodian, S. Jones, and B. Rogers. Springer, 2004, pp. 50–59.

[Bed11] B. B. Bederson. “The Promise of Zoomable User Interfaces”. In: *Behaviour & Information Technology* 30.6 (2011), pp. 853–866.

[Ber83] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.

[Bez+10] A. Bezerianos, F. Chevalier, P. Dragicevic, N. Elmqvist, and J. D. Fekete. “Graphdice: A System for Exploring Multivariate Social Networks”. In: *Proc. of Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*. John Wiley & Sons, Inc., 2010, pp. 863–872.

[BF00] B. A. Bell and S. K. Feiner. “Dynamic Space Management for User Interfaces”. In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 2000, pp. 239–248.

[BHJ09] M. Bastian, S. Heymann, and M. Jacomy. “Gephi: An Open Source Software for Exploring and Manipulating Networks.” In: *Proc. of International Conference on Weblogs and Social Media*. The AAAI Press, 2009.

[BHR14] S. Butscher, K. Hornbæk, and H. Reiterer. “SpaceFold and PhysicLenses: Simultaneous Multifocus Navigation on Touch Surfaces”. In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI)*. ACM, 2014, pp. 209–216.

[Bie+93] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. “Toolglass and Magic Lenses: The See-Through Interface”. In: *Proc. of Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH).* ACM, 1993, pp. 73–80.

[Bie+94] E. A. Bier, M. C. Stone, K. P. Fishkin, W. Buxton, and T. Baudel. “A Taxonomy of See-Through Tools”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1994, pp. 358–364.

[BN01] T. Barlow and P. Neville. “A Comparison of 2-D Visualizations of Hierarchies”. In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* IEEE Computer Society, 2001, pp. 131–139.

[Bos15a] M. Bostock. *Force-Directed Graph Layout in D3.* `http://bl.ocks.org/mbostock/4062045`. July 2015.

[Bos15b] M. Bostock. *Les Miserables Co-occurrence.* `http://bost.ocks.org/mike/miserables/`. July 2015.

[Bos15c] M. Bostock. *Treemap.* `http://bl.ocks.org/mbostock/4063582`. July 2015.

[BR03] P. Baudisch and R. Rosenholtz. “Halo: A Technique for Visualizing Off-Screen Objects”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2003, pp. 481–488.

[Bra94] A. Brandstädt. *Graphen und Algorithmen.* Vieweg+Teubner Verlag, 1994.

[BRL09] E. Bertini, M. Rigamonti, and D. Lalanne. “Extended Excentric Labeling”. In: *Computer Graphics Forum* 28.3 (2009), pp. 927–934.

[BRT95] L. Bergman, B. Rogowitz, and L. Treinish. “A Rule-based Tool for Assisting Colormap Selection”. In: *Visualization.* 1995, pp. 118–125.

[BSP97] E. Bier, M. Stone, and K. Pier. “Enhanced Illustration Using Magic Lens Filters”. In: *Computer Graphics and Applications* 17.6 (1997), pp. 62–70.

[Bur+12] M. Burch, C. Vehlow, N. Konevtsova, and D. Weiskopf. “Evaluating Partially Drawn Links for Directed Graph Edges”. In: *Proc. of International Conference on Graph Drawing.* Springer, 2012, pp. 226–237.

[Cal+88] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. “An Empirical Comparison of Pie vs. Linear Menus”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1988, pp. 95–100.

[CC13] M.-W. Chang and C. Collins. “Exploring Entities in Text with Descriptive Non-Photorealistic Rendering”. In: *Proc. of IEEE Pacific Visualization Symposium (PacificVis).* IEEE Computer Society, 2013, pp. 9–16.

[CCF97] M. Carpendale, D. Cowperthwaite, and F. Fracchia. “Extending Distortion Viewing from 2D to 3D”. In: *Computer Graphics and Applications* 17.4 (1997), pp. 42–51.

[CGH03] Q. Chen, J. Grundy, and J. Hosking. “An e-Whiteboard Application to Support Early Design-stage Sketching of UML Diagrams”. In: *Proc. of Symposium on Human Centric Computing Languages and Environments.* IEEE Computer Society, 2003, pp. 219–226.

[Che73] H. Chernoff. “The Use of Faces to Represent Points in K-Dimensional Space Graphically”. English. In: *Journal of the American Statistical Association* 68.342 (1973), pp. 361–368.

[CLP04] S. Carpendale, J. Ligh, and E. Pattison. “Achieving Higher Magnification in Context”. In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST).* ACM, 2004, pp. 71–80.

[CM01] M. S. T. Carpendale and C. Montagnese. “A Framework for Unifying Presentation Space”. In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST).* ACM, 2001, pp. 61–70.

[CM88] W. C. Cleveland and M. E. McGill. *Dynamic Graphics for Statistics.* CRC Press, Inc., 1988.

[CMS94] P. Cignoni, C. Montani, and R. Scopigno. “MagicSphere: an Insight Tool for 3D Data Visualization”. In: *Computer Graphics Forum* 13.3 (1994), pp. 317–328.

[CMS99] S. Card, J. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think.* Morgan Kaufmann, 1999.

[CR98] E. H. Chi and J. Riedl. “An Operator Interaction Framework for Visualization Systems”. In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* IEEE Computer Society, 1998, pp. 63–70.

[Crn+11] T. Crnovrsanin, I. Liao, Y. Wu, and K.-L. Ma. “Visual Recommendations for Network Navigation”. In: *Computer Graphics Forum* 30.3 (2011), pp. 1081–1090.

[CS04] H. Chen and B. M. Sharp. “Content-rich Biological Network Constructed by Mining PubMed Abstracts”. In: *BMC Bioinformatics* 5.1, 147 (2004).

[DG92] D. L. Donoho and M. Gasko. “Breakdown Properties of Location Estimates Based on Halfspace Depth and Projected Outlyingness”. In: *The Annals of Statistics* 20.4 (1992), pp. 1803–1827.

[DHT00] C. H. Damm, K. M. Hansen, and M. Thomsen. “Tool Support for Cooperative Object-oriented Design: Gesture Based Modelling on an Electronic Whiteboard”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2000, pp. 518–525.

[Die06] R. Diestel. *Graphentheorie (3. Aufl.)* Springer, 2006.

[Dix+97] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-computer Interaction.* Prentice-Hall, Inc., 1997.

[DMC15] M. Dumas, M. McGuffin, and P. Chasse. “VectorLens: Angular Selection of Curves within 2D Dense Visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 21 (2015), pp. 402–412.

[DPS02] J. Díaz, J. Petit, and M. Serna. “A Survey of Graph Layout Problems”. In: *ACM Computing Surveys* 34.3 (2002), pp. 313–356.

[DS13] C. Dunne and B. Shneiderman. “Motif Simplification: Improving Network Visualization Readability with Fan, Connector, and Clique Glyphs”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2013, pp. 3247–3256.

[DSA15] A. Debiasi, B. Simões, and R. D. Amicis. “3DArcLens: Interactive Network Analysis on Geographic Surfaces”. In: *Proc. of International Conference on Information Visualization Theory and Applications.* Scitepress, 2015, pp. 291–299.

[Dua+14] F. Duarte, F. Sikansi, F. Fatore, S. Fadel, and F. Paulovich. “Nmap: A Novel Neighborhood Preservation Space-filling Algorithm”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2063–2071.

[Ead84] P. Eades. “A Heuristic for Graph Drawing”. In: *Congressus Numerantium* 42 (1984). Ed. by D. S. Meek and v. G. H. J. Rees, pp. 149–160.

[EBD05] G. Ellis, E. Bertini, and A. Dix. “The Sampling Lens: Making Sense of Saturated Visualisations”. In: *CHI Extended Abstracts on Human Factors in Computing Systems.* ACM, 2005, pp. 1351–1354.

[ED06a] G. Ellis and A. J. Dix. “Enabling Automatic Clutter Reduction in Parallel Coordinate Plots”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 717–724.

[ED06b] G. Ellis and A. J. Dix. “The Plot, the Clutter, the Sampling and its Lens: Occlusion Measures for Automatic Clutter Reduction”. In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI).* ACM, 2006, pp. 266–269.

[EDF11] N. Elmqvist, P. Dragicevic, and J.-D. Fekete. “Color Lens: Adaptive Color Scale Optimization for Visual Exploration”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.6 (2011), pp. 795–807.

[EF10] N. Elmqvist and J.-D. Fekete. “Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines”. In: *IEEE Transactions on Visualization and Computer Graphics* 16.3 (2010), pp. 439–454.

[EHS69] T. O. Ellis, J. F. Heafner, and W. L. Sibley. *The Grail Project: An Experiment in Man-Machine Communications.* 1969.

[Eic+15] C. Eichner, T. Nocke, H.-J. Schulz, and H. Schumann. “Interactive Presentation of Geo-Spatial Climate Data in Multi-Display Environments”. In: *International Journal of Geo-Information (ISPRS)* 4.2 (2015), p. 493.

[Elm+08] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J. Fekete. “ZAME: Interactive Large-Scale Graph Visualization”. In: *Visualization Symposium*. Mar. 2008, pp. 215–222.

[Elm+11] N. Elmqvist, A. Van de Moere, H.-C. Jetter, D. Cernea, H. Reiterer, and T. J. Jankun-Kelly. “Fluid Interaction for Information Visualization”. In: *Information Visualization* 10.4 (2011), pp. 327–340.

[EMC15] EMC Corporation. *Digital Universe Invaded By Sensors*. `http://www.emc.com/about/news/press/2014/20140409-01.htm`. June 2015.

[EW14] S. van den Elzen and J. J. van Wijk. “Multivariate Network Exploration and Presentation: From Detail to Overview via Selections and Aggregations”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2310–2319.

[FD13] M. Frisch and R. Dachselt. “Visualizing Offscreen Elements of Node-Link Diagrams”. In: *Information Visualization* 12.2 (2013), pp. 133–162.

[FG98] A. Fuhrmann and E. Gröller. “Real-time Techniques for 3D Flow Visualization”. In: *Proc. of IEEE Visualization Conference (Vis)*. IEEE Computer Society, 1998, pp. 305–312.

[FHD09] M. Frisch, J. Heydekorn, and R. Dachselt. “Investigating Multi-touch and Pen Gestures for Diagram Editing on Interactive Surfaces”. In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS)*. ACM, 2009, pp. 149–156.

[FHD10] M. Frisch, J. Heydekorn, and R. Dachselt. “Diagram Editing on Interactive Displays Using Multi-touch and Pen Gestures”. In: *Proc. of International Conference on Diagrammatic Representation and Inference*. Springer, 2010, pp. 182–196.

[FHI89] M. Frigge, D. C. Hoaglin, and B. Iglewicz. “Some Implementations of the Box Plot”. In: *The American Statistician* 43.1 (1989), pp. 50–54.

[Fit54] P. Fitts. “The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement”. In: *Journal of Experimental Psychology* 47 (1954), pp. 381–391.

[Fox98] D. Fox. “Composing Magic Lenses”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1998, pp. 519–525.

[FP99] J.-D. Fekete and C. Plaisant. “Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1999, pp. 512–519.

[FR91] T. M. J. Fruchterman and E. M. Reingold. “Graph Drawing by Force-directed Placement”. In: *Software – Practice and Experience* 21.11 (1991), pp. 1129–1164.

[FS05] C. Forlines and C. Shen. “DTLens: Multi-user Tabletop Spatial Data Exploration”. In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST).* ACM, 2005, pp. 119–122.

[FS92] W. Felger and F. Schröder. “The Visualization Input Pipeline - Enabling Semantic Interaction in Scientific Visualization”. In: *Computer Graphics Forum* 11.3 (1992), pp. 139–151.

[FS95] K. P. Fishkin and M. C. Stone. “Enhanced Dynamic Queries via Movable Filters”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1995, pp. 415–420.

[Fur86] G. W. Furnas. “Generalized Fisheye Views”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1986, pp. 16–23.

[Gas+11] R. Gasteiger, M. Neugebauer, O. Beuing, and B. Preim. “The FLOWLENS: A Focus-and-Context Visualization Approach for Exploration of Blood Flow in Cerebral Aneurysms”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2183–2192.

[GFC04] M. Ghoniem, J.-D. Fekete, and P. Castagliola. “A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations”. In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* IEEE Computer Society, 2004, pp. 17–24.

[GFV13] H. Gibson, J. Faith, and P. Vickers. “A Survey of Two-Dimensional Graph Layout Techniques for Information Visualisation.” In: *Information Visualization* 12.3-4 (2013), pp. 324–357.

[GH07] J. Grundy and J. Hosking. “Supporting Generic Sketching-Based Input of Diagrams in a Domain-Specific Visual Language Meta-Tool”. In: *Proc. of International Conference on Software Engineering.* IEEE Computer Society, 2007, pp. 282–291.

[Gho+15] M. Ghoniem, M. Cornil, B. Broeksema, M. Stefas, and B. Otjacques. “Weighted Maps: Treemap Visualization of Geolocated Quantitative Data”. In: *Visualization and Data Analysis.* Ed. by D. L. Kao, M. C. Hao, M. A. Livingston, and T. Wischgoll. SPIE, 2015.

[Gla+14] S. Gladisch, H. Schumann, M. Ernst, G. Füllen, and C. Tominski. “Semi-Automatic Editing of Graphs with Customized Layouts”. In: *Computer Graphics Forum* 33.3 (2014), pp. 381–390.

[Gla+15a] S. Gladisch, H. Schumann, M. Luboschik, and C. Tominski. *Toward Using Matrix Visualizations for Graph Editing.* Poster at IEEE Conference on Information Visualization (InfoVis). 2015.

[Gla+15b] S. Gladisch, U. Kister, C. Tominski, R. Dachselt, and H. Schumann. *Mapping Tasks to Interactions for Graph Exploration and Graph Editing.* Poster at IEEE Conference on Information Visualization (InfoVis). 2015.

[Gla+15c] S. Gladisch, U. Kister, C. Tominski, R. Dachselt, and H. Schumann. *Mapping Tasks to Interactions for Graph Exploration and Graph Editing on Interactive Surfaces.* Tech. rep. University of Rostock, Technische Universität Dresden, 2015.

[Gou+12] L. Gou, X. Zhang, A. Luo, and P. Anderson. "SocialNetSense: Supporting sensemaking of social and structural features in networks with interactive visualization". In: *Proc. of IEEE Symposium on Visual Analytics Science and Technology (VAST).* 2012, pp. 133–142.

[GST13] S. Gladisch, H. Schumann, and C. Tominski. "Navigation Recommendations for Exploring Hierarchical Graphs". In: *Advances in Visual Computing, Proceedings of the International Symposium on Visual Computing (ISVC).* Ed. by G. Bebis, R. Boyle, B. Parvin, D. Koracin, B. Li, F. Porikli, V. Zordan, J. Klosowski, S. Coquillart, X. Luo, M. Chen, and D. Gotz. Vol. 8034. Springer, 2013, pp. 36–47.

[GT14] S. Gladisch and C. Tominski. *Toward Integrated Exploration and Manipulation of Data Attributes in Graphs.* Poster at IEEE Conference on Information Visualization (InfoVis). 2014.

[Gus+08] S. Gustafson, P. Baudisch, C. Gutwin, and P. Irani. "Wedge: Clutter-Free Visualization of Off-Screen Locations". In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2008, pp. 787–796.

[GW00] F. Guimbretiére and T. Winograd. "FlowMenu: Combining Command, Text, and Data Entry". In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST).* ACM, 2000, pp. 213–216.

[Hac05] S. Hachul. "A Potential-Field-Based Multilevel Algorithm for Drawing Large Graphs". PhD thesis. Universität zu Köln, 2005.

[Had14] S. Hadlak. "Graph Visualization in Space and Time". PhD thesis. University of Rostock, 2014.

[Hal+08] H. Halpin, D. Zielinski, R. Brady, and G. Kelly. "Redgraph: Navigating Semantic Web Networks using Virtual Reality". In: *Virtual Reality Conference.* Mar. 2008, pp. 257–258.

[Hal70] K. M. Hall. "An r-Dimensional Quadratic Placement Algorithm". In: *Management Science* 17.3 (1970), pp. 219–229.

[Ham03] F. van Ham. "Using Multilevel Call Matrices in Large Software Projects". In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* IEEE Computer Society, 2003, pp. 227–232.

[HB03] M. Harrower and C. A. Brewer. "ColorBrewer.org: An Online Tool for Selecting Color Schemes for Maps. The Cartographic". In: *The Carthographic Journal* 40.1 (2003), pp. 27–37.

[HD06] T. Hammond and R. Davis. "Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams". In: *ACM SIGGRAPH Courses.* ACM, 2006.

[HEH09] W. Huang, P. Eades, and S.-H. Hong. "Measuring Effectiveness of Graph Visualizations: A Cognitive Load Perspective". In: *Information Visualization* 8.3 (2009), pp. 139–152.

[Hei+04] R. Heilmann, D. Keim, C. Panse, and M. Sips. "RecMap: Rectangular Map Approximations". In: *Proc. of International Conference Information Visualisation (IV).* 2004, pp. 33–40.

[Hei+11] P. Heim, S. Lohmann, D. Tsendragchaa, and T. Ertl. "SemLens: Visual Analysis of Semantic Data with Scatter Plots and Semantic Lenses". In: *Proc. of International Conference on Semantic Systems.* ACM, 2011, pp. 175–178.

[Hen+12] N. Henry Riche, T. Dwyer, B. Lee, and S. Carpendale. "Exploring the Design Space of Interactive Link Curvature in Network Diagrams". In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI).* ACM, 2012, pp. 506–513.

[Hen08] N. Henry. "Exploring Large Social Networks with Matrix-Based Representations". PhD thesis. Cotutelle Université Paris-Sud and University of Sydney, 2008.

[HEW98] M. L. Huang, P. Eades, and J. Wang. "On-line Animated Visualization of Huge Graphs Using a Modified Spring Algorithm". In: *Journal of Visual Languages & Computing* 9.6 (1998), pp. 623–645.

[HF06] N. Henry and J.-D. Fekete. "MatrixExplorer: A Dual-Representation System to Explore Social Networks". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), pp. 677–684.

[HF07] N. Henry and J.-D. Fekete. "MatLink: Enhanced Matrix Visualization for Analyzing Social Networks". In: *Proc. of International Conference on Human-computer Interaction.* Springer, 2007, pp. 288–302.

[HFM07] N. Henry, J.-D. Fekete, and M. J. McGuffin. "NodeTrix: A Hybrid Visualization of Social Networks". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1302–1309.

[HHE06] W. Huang, S.-H. Hong, and P. Eades. "Layout Effects on Sociogram Perception". In: *Proc. of International Conference on Graph Drawing.* Springer, 2006, pp. 262–273.

[HK04] D. Harel and Y. Koren. "Graph Drawing by High-Dimensional Embedding". In: *Journal of Graph Algorithms and Applications* 8.2 (2004), pp. 195–214.

[HM90] R. B. Haber and D. A. Mcnabb. “Visualization Idioms: A Conceptual Model for Scientific Visualization Systems”. In: *Visualization in Scientific Computing*. Ed. by B. Shriver, G. Nielson, and L. Rosenblum. IEEE Computer Society, 1990, pp. 74–93.

[HMM00] I. Herman, G. Melançon, and M. S. Marshall. “Graph Visualization and Navigation in Information Visualization: a Survey”. In: *IEEE Transactions on Visualization and Computer Graphics* 6.1 (2000), pp. 24–43.

[HNR68] P. Hart, N. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[Hol06] D. Holten. “Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 741–748.

[HP09] F. van Ham and A. Perer. “Search, Show Context, Expand on Demand: Supporting Large Graph Exploration with Degree-of-Interest”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 953–960.

[HP11] J. Heer and A. Perer. “Orion: A System for Modeling, Transformation and Visualization of Multidimensional Heterogeneous Networks”. In: *Proc. of IEEE Symposium on Visual Analytics Science and Technology (VAST)*. 2011, pp. 51–60.

[HSD09] F. Ham, H.-J. Schulz, and J. M. Dimicco. “Honeycomb: Visual Analysis of Large Scale Social Networks”. In: *Proc. of International Conference on Human-Computer Interaction*. Springer, 2009, pp. 429–442.

[HTE11] C. Hurter, A. Telea, and O. Ersoy. “MoleView: An Attribute and Structure-Based Semantic Lens for Large Element-Based Plots”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2600–2609.

[HVW05] D. Holten, R. Vliegen, and J. van Wijk. “Visual Realism for the Visualization of Software Metrics”. In: *Visualizing Software for Understanding and Analysis*. 2005, pp. 1–6.

[HW02] F. van Ham and J. J. van Wijk. “Beamtrees: Compact Visualization of Large Hierarchies”. In: *Proc. of IEEE Symposium Information Visualization (InfoVis)*. IEEE Computer Society, 2002, pp. 93–100.

[HW04] F. van Ham and J. van Wijk. “Interactive Visualization of Small World Graphs”. In: *Proc. of IEEE Symposium Information Visualization (InfoVis)*. IEEE Computer Society, 2004, pp. 199–206.

[HW09] D. Holten and J. J. van Wijk. “A User Study on Visualizing Directed Edges in Graphs”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 2009, pp. 2299–2308.

[ID90] A. Inselberg and B. Dimsdale. “Parallel Coordinates: A Tool for Visualizing Multi-dimensional Geometry”. In: *Proc. of Conference on Visualization.* IEEE Computer Society, 1990, pp. 361–378.

[IU97] H. Ishii and B. Ullmer. “Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1997, pp. 234–241.

[JD13] Y. Jansen and P. Dragicevic. “An Interaction Model for Visualizations Beyond The Desktop”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2396–2405.

[JDK10] I. Jusufi, Y. Dingjie, and A. Kerren. “The Network Lens: Interactive Exploration of Multivariate Networks Using Visual Filtering”. In: *Proc. of International Conference Information Visualisation (IV).* 2010, pp. 35–42.

[JKZ13] I. Jusufi, A. Kerren, and B. Zimmer. “Multivariate Network Exploration with JauntyNets”. In: *Proc. of International Conference Information Visualisation (IV).* 2013, pp. 19–27.

[Joh93] B. S. Johnson. “Treemaps: Visualizing hierarchical and categorical data”. PhD thesis. University of Maryland, 1993.

[JS91] B. Johnson and B. Shneiderman. “Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures”. In: *Proc. of Conference on Visualization.* IEEE Computer Society, 1991, pp. 284–291.

[Jus+12] I. Jusufi, C. Klukas, A. Kerren, and F. Schreiber. “Guiding the Interactive Exploration of Metabolic Pathway Interconnections”. In: *Information Visualization* 11.2 (2012), pp. 136–150.

[Jus+14] I. Jusufi, A. Kerren, J. Liu, and B. Zimmer. “Visual Exploration of Relationships between Document Clusters”. In: *Information Visualization Theory and Applications (IVAPP).* 2014, pp. 195–203.

[Kan+11] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono. “Research Directions in Data Wrangling: Visualizations and Transformations for Usable and Credible Data”. In: *Information Visualization* 10 (2011), pp. 271–288.

[Kar+10] P. Karnick, D. Cline, S. Jeschke, A. Razdan, and P. Wonka. “Route Visualization Using Detail Lenses”. In: *IEEE Transactions on Visualization and Computer Graphics* 16.2 (2010), pp. 235–247.

[KB94] G. Kurtenbach and W. Buxton. “User Learning and Performance with Marking Menus”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1994, pp. 258–264.

[KCH03] Y. Koren, L. Carmel, and D. Harel. “Drawing Huge Graphs by Algebraic Multigrid Optimization”. In: *Multiscale Modeling and Simulation* 1.4 (2003), pp. 645–673.

[Kea98] A. Keahey. "The Generalized Detail-In-Context Problem". In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* IEEE Computer Society, 1998, pp. 44–51.

[KEC06] R. Keller, C. M. Eckert, and P. J. Clarkson. "Matrices or Node-link Diagrams: Which Visual Representation is Better for Visualising Connectivity Models?" In: *Information Visualization* 5.1 (Mar. 2006), pp. 62–76.

[Kei+05] D. A. Keim, F. Mansmann, C. Panse, J. Schneidewind, and M. Sips. "Mail Explorer - Spatial and Temporal Exploration of Electronic Mail". In: *Proc. of Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym).* Eurographics Association, 2005, pp. 247–254.

[Kei02] D. A. Keim. "Information Visualization and Visual Data Mining". In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (Jan. 2002), pp. 1–8.

[Kel+12] T. Kelder, M. P. van Iersel, K. Hanspers, M. Kutmon, B. R. Conklin, C. T. A. Evelo, and A. R. Pico. "WikiPathways: Building Research Communities on Biological Pathways". In: *Nucleic Acids Research* 40 (2012), pp. 1301–1307.

[Kin10] R. Kincaid. "SignalLens: Focus+Context Applied to Electronic Time Series". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 900–907.

[Kis+15] U. Kister, P. Reipschläger, F. Matulic, and R. Dachselt. "BodyLenses – Embodied Magic Lenses and Personal Territories for Wall Displays". In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS).* ACM, Nov. 2015.

[KL83] J. B. Kruskal and J. M. Landwehr. "Icicle Plots: Better Displays for Hierarchical Clustering". English. In: *The American Statistician* 37.2 (1983), pp. 162–168.

[KMS09] D. Kalkofen, E. Méndez, and D. Schmalstieg. "Comprehensible Visualization for Augmented Reality". In: *IEEE Transactions on Visualization and Computer Graphics* 15.2 (2009), pp. 193–204.

[KMT12] A. Kobayashi, K. Misue, and J. Tanaka. "Edge Equalized Treemaps". In: *Proc. of International Conference Information Visualisation (IV).* 2012, pp. 7–12.

[Knu09] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing.* 1st. Addison-Wesley Professional, 2009.

[Knu15] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing.* `http://www-cs-faculty.stanford.edu/~uno/sgb.html`. Oct. 2015.

[KPW14] A. Kerren, H. C. Purchase, and M. O. Ward, eds. *Multivariate Network Visualization - Dagstuhl Seminar #13201.* Vol. 8380. Springer, 2014.

[KRD14] U. Kister, P. Reipschläger, and R. Dachselt. “Multi-Touch Manipulation of Magic Lenses for Information Visualization”. In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS).* ACM, 2014, pp. 431–434.

[Krü+13] R. Krüger, D. Thom, M. Wörner, H. Bosch, and T. Ertl. “TrajectoryLenses - A Set-based Filtering and Exploration Technique for Long-term Trajectory Data”. In: *Computer Graphics Forum* 32.3 (2013), pp. 451–460.

[KSW06] J. H. Krüger, J. Schneider, and R. Westermann. “ClearView: An Interactive Context Preserving Hotspot Visualization Technique”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 941–948.

[Kur+99] G. Kurtenbach, G. W. Fitzmaurice, R. N. Owen, and T. Baudel. “The Hotbox: Efficient Access to a Large Number of Menu-items”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1999, pp. 231–237.

[LA94] Y. K. Leung and M. D. Apperley. “A Review and Taxonomy of Distortion-oriented Presentation Techniques”. In: *ACM Transactions on Computer-Human Interaction* 1.2 (1994), pp. 126–160.

[Lam08] H. Lam. “A Framework of Interaction Costs in Information Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1149–1156.

[Lan+11] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. W. Fellner. “Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges”. In: *Computer Graphics Forum* 30.6 (2011), pp. 1719–1749.

[Lee+06] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. “Task Taxonomy for Graph Visualization”. In: *Proc. of AVI Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization.* ACM, 2006, pp. 1–5.

[LGB07] J. Looser, R. Grasset, and M. Billinghurst. “A 3D Flexible and Tangible Magic Lens in Augmented Reality”. In: *Proc. of IEEE International Symposium on Mixed and Augmented Reality (ISMAR).* IEEE Computer Society, 2007, pp. 51–54.

[LGF10] G. J. Lepinski, T. Grossman, and G. Fitzmaurice. “The Design and Evaluation of Multitouch Marking Menus”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2010, pp. 2233–2242.

[LHJ01] E. LaMar, B. Hamann, and K. Joy. “A Magnification Lens for Interactive Volume Visualization”. In: *Proc. of Pacific Conference on Computer Graphics and Applications.* IEEE Computer Society, 2001, pp. 223–232.

[LK01] M. Last and A. Kandel. “Automated Detection of Outliers in Real-World Data”. In: *Proc. of International Conference on Intelligent Technologies.* 2001, pp. 292–301.

[LRW10] U. Laufs, C. Ruff, and A. Weisbecker. “MT4j – an Open Source Platform for Multi-Touch Software Development”. In: *VIMation* 1 (2010), pp. 58–64.

[LSC08] M. Luboschik, H. Schumann, and H. Cords. “Particle-Based Labeling: Fast Point-Feature Labeling without Obscuring Other Visual Features”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1237–1244.

[LWG11] C. Lambeck, J. Wojdziak, and R. Groh. “Facet Lens: Local Exploration and Discovery in Globally Faceted Data Sets”. In: *Proc. of Conference on Creativity and Innovation in Design.* ACM, 2011, pp. 85–88.

[Mac86] J. Mackinlay. “Automating the Design of Graphical Presentations of Relational Information”. In: *ACM Transactions on Graphics* 5.2 (1986), pp. 110–141.

[Mad+05] J. Madadhain, D. Fisher, P. Smyth, S. White, and Y. Boey. “Analysis and visualization of network data using JUNG”. In: *Journal of Statistical Software* 10 (2005), pp. 1–35.

[Man+07] F. Mansmann, D. Keim, S. North, B. Rexroad, and D. Sheleheda. “Visual Analysis of Network Traffic for Resource Planning, Interactive Monitoring, and Interpretation of Security Threats”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1105–1112.

[Mat+03] O. Mattausch, T. Theußl, H. Hauser, and E. Gröller. “Strategies for Interactive Exploration of 3D Flow Using Evenly-spaced Illuminated Streamlines”. In: *Proc. of Spring Conference on Computer Graphics.* ACM, 2003, pp. 213–222.

[Mcg+02] M. Mcguffin, N. Burtnyk, G. Kurtenbach, and K. Street. “FaST Sliders: Integrating marking menus and the adjustment of continuous values. Graphics Interface”. In: *Graphics Interface Conference Proceedings.* 2002, pp. 35–42.

[ME09] B. McDonnel and N. Elmqvist. “Towards utilizing GPUs in information visualization: A model and implementation of image-space operations”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1105–1112.

[MJ09] M. J. Mcguffin and I. Jurisica. “Interaction Techniques for Selecting and Manipulating Subgraphs in Network Visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 937–944.

[MML07] C. Mueller, B. Martin, and A. Lumsdaine. “A comparison of vertex ordering algorithms for large graph visualization”. In: *Visualization.* Feb. 2007, pp. 141–148.

[Mos+09] T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J.-D. Fekete. “Topology-aware Navigation in Large Networks”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2009, pp. 2319–2328.

[MSK12] T. May, M. Steiger, and J. D. J. Kohlhammer. “Using Signposts for Navigation in Large Graphs”. In: *Computer Graphics Forum* 31 (2012), pp. 985–994.

[MSW14] K. Marriott, P. J. Stuckey, and M. Wybrow. “Seeing Around Corners: Fast Orthogonal Connector Routing”. In: *Diagrammatic Representation and Inference.* 2014, pp. 31–37.

[Mül+14] J. Müller, T. Schwarz, S. Butscher, and H. Reiterer. “Back to Tangibility: A Post-Wimp Perspective on Control Room Design”. In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI).* IEEE Computer Society, 2014, pp. 57–64.

[Mun14] T. Munzner. *Visualization Analysis and Design.* A K Peters, 2014.

[MW14] M. E. Mott and J. O. Wobbrock. “Beating the Bubble: Using Kinematic Triggering in the Bubble Lens for Acquiring Small, Dense Targets”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2014, pp. 733–742.

[Nor13] D. A. Norman. *The Design of Everyday Things.* Basic Books, 2013.

[Pan+11] A. Panagiotidis, H. Bosch, S. Koch, and T. Ertl. “EdgeAnalyzer: Exploratory Analysis through Advanced Edge Interaction”. In: *Proc. of Hawaii International Conference on System Sciences.* IEEE Computer Society, 2011.

[PD15] B. Preim and R. Dachselt. *Interaktive Systeme.* Vol. 2. Springer, 2015.

[PGB02] C. Plaisant, J. Grosjean, and B. Bederson. “SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation”. In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* IEEE Computer Society, 2002, pp. 57–64.

[PH11] A. Perer and F. van Ham. *Integrating Querying and Browsing in Partial Graph Visualizations.* Tech. rep. IBM Research, 2011.

[Pin+12] C. Pindat, E. Pietriga, O. Chapuis, and C. Puech. “JellyLens: Content-aware Adaptive Lenses”. In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST).* ACM, 2012, pp. 261–270.

[Pin+13] C. Pindat, E. Pietriga, O. Chapuis, and C. Puech. “Drilling into Complex 3D Models with Gimlenses”. In: *Proc. of Symposium on Virtual Reality Software and Technology.* ACM, 2013, pp. 223–230.

[Poo+00] S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot. “Control Menus: Excecution and Control in a Single Interactor”. In: *CHI Extended Abstracts on Human Factors in Computing Systems.* ACM, 2000, pp. 263–264.

[Pur02] H. C. Purchase. “Metrics for Graph Drawing Aesthetics”. In: *Journal of Visual Languages & Computing* 13.5 (2002), pp. 501–516.

[Pur98] H. C. Purchase. “Performance of Layout Algorithms: Comprehension, not Computation”. In: *Journal of Visual Languages & Computing* 9.6 (1998), pp. 647–657.

[PW08] A. J. Pretorius and J. J. van Wijk. “Visual Inspection of Multivariate Graphs”. In: *Proc. of Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym).* John Wiley & Sons, Inc., 2008, pp. 967–974.

[RC94] R. Rao and S. K. Card. “The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus + Context Visualization for Tabular Information”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 1994, pp. 318–322.

[RH04] T. Ropinski and K. Hinrichs. “Real-Time Rendering of 3D Magic Lenses Having Arbitrary Convex Shapes”. In: *Proc. of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision.* UNION Agency - Science Press, 2004, pp. 379–386.

[RHS05] T. Ropinski, K. H. Hinrichs, and F. Steinicke. “A Solution for the Focus and Context Problem in Geo-Virtual Environments”. In: *Proc. of ISPRS Workshop on Dynamic and Multi-dimensional GIS.* 2005, pp. 144–149.

[RLE05] K. Ryall, Q. Li, and A. Esenther. “Temporal Magic Lens: Combined Spatial and Temporal Query and Presentation”. In: *Proc. of Conference on Human-Computer Interaction.* Springer, 2005, pp. 809–822.

[RM93] G. G. Robertson and J. D. Mackinlay. “The Document Lens”. In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST).* ACM, 1993, pp. 101–108.

[RMF09] S. Rufiange, M. J. McGuffin, and C. Fuhrman. “Visualisation Hybride Des Liens Hierlarchiques Incorporant Des Treemaps Dans Une Matrice D’Adjacence”. In: *Proc. of International Conference on Association Francophone D’Interaction Homme-Machine.* ACM, 2009, pp. 51–54.

[Rob87] G. Robins. *The ISI grapher: a Portable Tool for Displaying Graphs Pictorially.* Nr. 196. University of Southern California, Information Sciences Institute, 1987.

[Rod+11] E. Rodrigues, N. Milic-Frayling, M. Smith, B. Shneiderman, and D. Hansen. “Group-in-a-Box Layout for Multi-faceted Analysis of Communities”. In: *Inernational Conference on Social Computing.* 2011, pp. 354–361.

[Rod15] K. Rodden. *Sequences Sunburst.* `http://bl.ocks.org/kerryrodden/7090426`. July 2015.

[Roh+10] M. Rohrschneider, C. Heine, A. Reichenbach, A. Kerren, and G. Scheuermann. "A Novel Grid-based Visualization Approach for Metabolic Networks with Advanced Focus & Context View". In: *Proceedings of the 17th International Conference on Graph Drawing*. Springer-Verlag, 2010, pp. 268–279.

[Roh+12] H. Rohn, A. Hartmann, A. Junker, B. Junker, and F. Schreiber. "FluxMap: a VANTED Add-on for the Visual Exploration of Flux Distributions in Biological Networks." In: *BMC Systems Biology* 6.33 (2012).

[Ros+04] G. E. Rosario, E. A. Rundensteiner, D. C. Brown, M. O. Ward, and S. Huang. "Mapping Nominal Values to Numbers for Effective Visualization". In: *Information Visualization* 3.2 (2004), pp. 80–95.

[SA06] B. Shneiderman and A. Aris. "Network Visualization by Semantic Substrates". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 733–740.

[SB04] S. dos Santos and K. Brodlie. "Gaining Understanding of Multivariate and Multidimensional Data through Visualization". In: *Computers & Graphics* 28.3 (2004), pp. 311–325.

[SB92] M. Sarkar and M. H. Brown. "Graphical Fisheye Views of Graphs". In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1992, pp. 83–91.

[SB94] M. Sarkar and M. H. Brown. "Graphical Fisheye Views". In: *Communications of the ACM* 37.12 (1994), pp. 73–83.

[Sch+08] H.-J. Schulz, M. John, A. Unger, and H. Schumann. "Visual Analysis of Bipartite Biological Networks". In: *Proc. of Eurographics Conference on Visual Computing for Biomedicine*. Eurographics Association, 2008, pp. 135–142.

[Sch+10] S. Schmidt, M. A. Nacenta, R. Dachselt, and S. Carpendale. "A Set of Multi-Touch Graph Interaction Techniques". In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS)*. ACM, 2010, pp. 113–116.

[Sch04] M. Schumaker. "Matrix Visualization of Graphs". PhD thesis. Rensselaer Polytechnic Institute, 2004.

[Sch11] H.-J. Schulz. "Treevis.net: A Tree Visualization Reference". In: *Computer Graphics and Applications* 31.6 (Nov. 2011), pp. 11–15.

[SFB94] M. C. Stone, K. P. Fishkin, and E. A. Bier. "The Movable Filter as a User Interface Tool". In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1994, pp. 306–312.

[SFR01] S. L. Stoev, M. Feurer, and M. Ruckaberle. "Exploring the Past: A Toolset for Visualization of Historical Events in Virtual Environments". In: *Proc. of Symposium on Virtual Reality Software and Technology*. ACM, 2001, pp. 63–70.

[Sha+03] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. “Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks”. In: *Genome Research* 13.11 (2003), pp. 2498–2504.

[Sha+99] C. D. Shaw, J. A. Hall, D. S. Ebert, and D. A. Roberts. “Interactive Lens Visualization Techniques”. In: *Proc. of IEEE Visualization Conference (Vis)*. IEEE Computer Society, 1999, pp. 155–160.

[Shi+14] L. Shi, Q. Liao, H. Tong, Y. Hu, Y. Zhao, and C. Lin. “Hierarchical Focus+Context Heterogeneous Network Visualization”. In: *Proc. of IEEE Pacific Visualization Symposium (PacificVis)*. 2014, pp. 89–96.

[Shn83] B. Shneiderman. “Direct Manipulation: A Step Beyond Programming Languages”. In: *Computer* 16.8 (1983), pp. 57–69.

[Shn92] B. Shneiderman. “Tree Visualization with Tree-maps: 2-d Space-filling Approach”. In: *ACM Transactions on Graphics* 11.1 (Jan. 1992), pp. 92–99.

[Shn96] B. Shneiderman. “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations”. In: *Proc. of Symposium on Visual Languages*. IEEE Computer Society, 1996, pp. 336–343.

[SHQ08] R. Shannon, T. Holland, and A. Quigley. *Multivariate Graph Drawing using Parallel Coordinate Visualisations*. Tech. rep. UCD School of Computer Science and Informatics, 2008.

[SL10] F. Steinbrückner and C. Lewerentz. “Representing Development History in Software Cities”. In: *Proc. of International Symposium on Software Visualization*. ACM, 2010, pp. 193–202.

[SM07] Z. Sheny and K.-L. Maz. “Path Visualization for Adjacency Matrices”. In: *Proc. of Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*. Eurographics Association, 2007, pp. 83–90.

[Som+10] A. Som, C. Harder, B. Greber, M. Siatkowski, Y. Paudel, G. Warsow, C. Cap, H. Schöler, and G. Fuellen. “The PluriNetWork: An Electronic Representation of the Network Underlying Pluripotency in Mouse, and Its Applications”. In: *PLoS One* 5.12 (2010).

[Spe07] R. Spence. *Information Visualization: Design for Interaction (2Nd Edition)*. Prentice-Hall, Inc., 2007.

[Spi+10] M. Spindler, C. Tominski, H. Schumann, and R. Dachselt. “Tangible Views for Information Visualization”. In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS)*. ACM, 2010, pp. 157–166.

[SSD09] M. Spindler, S. Stellmach, and R. Dachselt. “PaperLens: Advanced Magic Lens Interaction Above the Tabletop”. In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS)*. ACM, 2009, pp. 69–76.

[Sta+00] J. Stasko, R. Catrambone, M. Guzial, and K. McDonald. "An Evaluation of Space-filling Information Visualizations for Depicting Hierarchical Structures". In: *International Journal of Human-Computer Studies* 53.5 (Nov. 2000), pp. 663–694.

[Sta+07] J. Stasko, C. Gorg, Z. Liu, and K. Singhal. "Jigsaw: Supporting Investigative Analysis through Interactive Visualization". In: *Proc. of IEEE Symposium on Visual Analytics Science and Technology (VAST)*. 2007, pp. 131–138.

[STS05] P. Schulze-Wollgast, C. Tominski, and H. Schumann. "Enhancing Visual Exploration by Appropriate Color Coding". In: *Proc. of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. 2005, pp. 203–210.

[SZ00] J. Stasko and E. Zhang. "Focus+context Display and Navigation Techniques for Enhancing Radial, Space-filling Hierarchy Visualizations". In: *Proc. of International Conference Information Visualisation (IV)*. 2000, pp. 57–65.

[Tam07] R. Tamassia. *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2007.

[TAS09] C. Tominski, J. Abello, and H. Schumann. "CGV – An Interactive Graph Visualization System". In: *Computers & Graphics* 33.6 (2009), pp. 660–678.

[Tel07] A. Telea. *Data Visualization: Principles and Practice*. A K Peters, 2007.

[Ter15] G. Terrill. *Neo4j - an Embedded, Network Database*. `http://www.infoq.com/news/2008/06/neo4j`. Accessed: 2015-01-12. 2015.

[TFJ12] C. Tominski, C. Forsell, and J. Johansson. "Interaction Support for Visual Comparison Inspired by Natural Behavior". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2719–2728.

[TFS08a] C. Thiede, G. Fuchs, and H. Schumann. "Smart Lenses". In: *Proc. of Smart Graphics (SG)*. Springer, 2008, pp. 178–189.

[TFS08b] C. Tominski, G. Fuchs, and H. Schumann. "Task-Driven Color Coding". In: *Proc. of International Conference Information Visualisation (IV)*. IEEE Computer Society, 2008, pp. 373–380.

[Tha+14] F. Thalmann, U. von Zadow, M. Heckel, and R. Dachselt. "X-O Arch Menu: Combining Precise Positioning with Efficient Menu Selection on Touch Devices". In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS)*. ACM, 2014, pp. 317–322.

[TN87] W. Tichy and F. Newbery. "Knowledge-based Editors for Directed Graphs". English. In: *European Software Engineering Conference*. Ed. by H. Nichols and D. Simpson. Vol. 289. Springer, 1987, pp. 99–109.

[Tom+06] C. Tominski, J. Abello, F. van Ham, and H. Schumann. "Fisheye Tree Views and Lenses for Graph Visualization". In: *Proc. of International Conference Information Visualisation (IV)*. IEEE Computer Society, 2006, pp. 17–24.

[Tom+12] C. Tominski, H. Schumann, G. Andrienko, and N. Andrienko. "Stacking-Based Visualization of Trajectory Attribute Data". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2565–2574.

[Tom+14] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann. "A Survey on Interactive Lenses in Visualization". In: *EuroVis State-of-the-Art Reports*. Eurographics Association, 2014, pp. 43–62.

[Tom06] C. Tominski. "Event-Based Visualization for User-Centered Visual Analysis". PhD thesis. University of Rostock, Germany, 2006.

[Tra+08] M. Trapp, T. Glander, H. Buchholz, and J. Dolner. "3D Generalization Lenses for Interactive Focus + Context Visualization of Virtual City Models". In: *Proc. of International Conference Information Visualisation (IV)*. IEEE Computer Society, 2008, pp. 356–361.

[UČK13] J. Ukrop, Z. Číková, and P. Kapec. "Visual Access to Graph Content Using Magic Lenses and Filtering". In: *Proc. of Spring Conference on Computer Graphics*. ACM, 2013, pp. 23–30.

[Via+10] C. Viau, M. J. McGuffin, Y. Chiricota, and I. Jurisica. "The FlowVizMenu and Parallel Scatterplot Matrix: Hybrid Multidimensional Visualizations for Network Exploration". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 1100–1108.

[Vie+96] J. Viega, M. Conway, G. H. Williams, and R. F. Pausch. "3D Magic Lenses". In: *Proc. of ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 1996, pp. 51–58.

[Voi+09] S. Voida, M. Tobiasz, J. Stromer, P. Isenberg, and S. Carpendale. "Getting Practical with Interactive Tabletop Displays: Designing for Dense Data, "Fat Fingers," Diverse Interactions, and Face-to-face Collaboration". In: *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS)*. ACM, 2009, pp. 109–116.

[VPF06] E. R. A. Valiati, M. S. Pimenta, and C. M. D. S. Freitas. "A Taxonomy of Tasks for Guiding the Evaluation of Multidimensional Visualizations". In: *Proc. of AVI Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization*. ACM, 2006, pp. 1–6.

[Wal14] B. Waldhauer. "Touch Interaction for Exploring and Manipulating Graphs". MA thesis. University of Rostock, 2014.

[Wan+05] L. Wang, Y. Zhao, K. Mueller, and A. E. Kaufman. "The Magic Volume Lens: An Interactive Focus+Context Technique for Volume Rendering". In: *Proc. of IEEE Visualization Conference (Vis)*. IEEE Computer Society, 2005, pp. 367–374.

[War04] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2004.

[Wat05] M. Wattenberg. "A Note on Space-filling Visualizations and Space-filling Curves". In: *Proc. of International Conference Information Visualisation (IV).* Oct. 2005, pp. 181–186.

[Wat06] M. Wattenberg. "Visual Exploration of Multivariate Graphs". In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, 2006, pp. 811–819.

[WCG03] N. Wong, M. S. T. Carpendale, and S. Greenberg. "EdgeLens: An Interactive Method for Managing Edge Congestion in Graphs". In: *Proc. of IEEE Symposium Information Visualization (InfoVis).* IEEE Computer Society, 2003, pp. 51–58.

[WD08] J. Wood and J. Dykes. "Spatially Ordered Treemaps". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1348–1355.

[Wet03] K. Wetzel. *Pebbles – using Circular Treemaps to visualize disk usage.* 2003.

[WL07] R. Wettel and M. Lanza. "Visualizing Software Systems as Cities". In: *Visualizing Software for Understanding and Analysis.* 2007, pp. 92–99.

[WMS05] M. Wybrow, K. Marriott, and P. J. Stuckey. "Incremental Connector Routing". In: *Graph Drawing.* 2005, pp. 446–457.

[WMS09] M. Wybrow, K. Marriott, and P. J. Stuckey. "Orthogonal Connector Routing". In: *Graph Drawing.* 2009, pp. 219–231.

[Wro10] L. Wroblewski. *Touch Target Sizes.* `http://www.lukew.com/ff/entry.asp?1085`. 2010.

[WS98] D. J. Watts and S. H. Strogatz. "Collective dynamics of'small-world'networks." In: *Nature* 393.6684 (1998), pp. 409–10.

[WT06] Y. Wu and M. Takatsuka. "Visualizing Multivariate Network on the Surface of a Sphere". In: *Proc. of Asia-Pacific Symposium on Information Visualisation - Volume 60.* Australian Computer Society, Inc., 2006, pp. 77–83.

[WT08] Y. Wu and M. Takatsuka. "Visualizing Multivariate Networks: A Hybrid Approach". In: *Visualization Symposium.* 2008, pp. 223–230.

[WTM06] Y. Wang, S. T. Teoh, and K.-L. Ma. "Evaluating the Effectiveness of Tree Visualization Systems for Knowledge Discovery". In: *Proc. of Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym).* Eurographics Association, 2006, pp. 67–74.

[Xu+07] K. Xu, A. Cunningham, S.-H. Hong, and B. Thomas. "GraphScape: Integrated Multivariate Network Visualization". In: *Visualization.* Feb. 2007, pp. 33–40.

[Xu+12] K. Xu, C. Rooney, P. Passmore, D.-H. Ham, and P. Nguyen. "A User Study on Curved Edges in Graph Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2449–2456.

[Yi+07] J. S. Yi, Y. a. Kang, J. Stasko, and J. Jacko. “Toward a Deeper Understanding of the Role of Interaction in Information Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1224–1231.

[ZCB11] J. Zhao, F. Chevalier, and R. Balakrishnan. “KronoMiner: Using Multi-foci Navigation for the Visual Exploration of Time-series Data”. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 2011, pp. 1737–1746.

[Zha+11] J. Zhao, F. Chevalier, E. Pietriga, and R. Balakrishnan. “Exploratory Analysis of Time-Series with ChronoLenses”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2422–2431.

[Zho+13] H. Zhou, P. Xu, X. Yuan, and H. Qu. “Edge Bundling in Information Visualization”. In: *Tsinghua Science and Technology* 18.2 (2013), pp. 145–156.

[ZK14] B. Zimmer and A. Kerren. *Applying Heat Maps in a Web-Based Collaborative Graph Visualization*. Poster Abstracts, IEEE Symposium Information Visualization (InfoVis). 2014.

[ZK15] B. Zimmer and A. Kerren. “Displaying User Behavior in the Collaborative Graph Visualization System OnGraX”. In: *Graph Drawing and Network Visualization*. Ed. by E. Di Giacomo and A. Lubiw. Vol. 9411. Springer International Publishing, 2015, pp. 247–259.

[ZMC05] S. Zhao, M. McGuffin, and M. Chignell. “Elastic Hierarchies: Combining Treemaps and Node-Link Diagrams”. In: *Proc. of International Conference Information Visualisation (IV)*. 2005, pp. 57–64.

List of Scientific Publications and Lectures

Journal Articles

[Gla+14] S. Gladisch, H. Schumann, M. Ernst, G. Füllen, and C. Tominski. "Semi-Automatic Editing of Graphs with Customized Layouts". In: *Computer Graphics Forum* 33.3 (2014), pp. 381–390.

A state-of-the-art report titled *"Interactive Lenses in Visualization: An Extended Survey"* was submitted to the journal *Computer Graphics Forum* and is currently reviewed.

Conference Publications

[Gla+15a] S. Gladisch, H. Schumann, M. Luboschik, and C. Tominski. *Toward Using Matrix Visualizations for Graph Editing.* Poster at IEEE Conference on Information Visualization (InfoVis). 2015.

[Gla+15b] S. Gladisch, U. Kister, C. Tominski, R. Dachselt, and H. Schumann. *Mapping Tasks to Interactions for Graph Exploration and Graph Editing.* Poster at IEEE Conference on Information Visualization (InfoVis). 2015.

[GST13] S. Gladisch, H. Schumann, and C. Tominski. "Navigation Recommendations for Exploring Hierarchical Graphs". In: *Advances in Visual Computing, Proceedings of the International Symposium on Visual Computing (ISVC).* Ed. by G. Bebis, R. Boyle, B. Parvin, D. Koracin, B. Li, F. Porikli, V. Zordan, J. Klosowski, S. Coquillart, X. Luo, M. Chen, and D. Gotz. Vol. 8034. Springer, 2013, pp. 36–47.

[GT14] S. Gladisch and C. Tominski. *Toward Integrated Exploration and Manipulation of Data Attributes in Graphs.* Poster at IEEE Conference on Information Visualization (InfoVis). 2014.

[Tom+14] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann. "A Survey on Interactive Lenses in Visualization". In: *EuroVis State-of-the-Art Reports.* Eurographics Association, 2014, pp. 43–62.

A paper with the title *"Visual Editing of Node Attribute Data of Multivariate Graphs"* was submitted to *EuroVis* conference 2016 and is currently undergoing the review process.

Miscellaneous Contributions

[Gla+15c] S. Gladisch, U. Kister, C. Tominski, R. Dachselt, and H. Schumann. *Mapping Tasks to Interactions for Graph Exploration and Graph Editing on Interactive Surfaces*. Tech. rep. University of Rostock, Technische Universität Dresden, 2015.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die eingereichte Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Diese Arbeit hat noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen.

Rostock, 29.01.2016

. . . .

Stefan Gladisch

Thesis Statements

1. Visualization approaches are the established means to support the user in extracting useful information from graphs. Although such information can reveal the need for editing a graph and can provide insight about the effect of previously applied data changes, graph editing is still mostly carried out by using external tools. This separate approach complicates the user's work.

2. Existing literature provides first approaches for directly editing graphs in visual representations of the data. However, these approaches focus on editing the structure of rather small graphs and are limited to node-link representations. The structural editing of larger graphs with established and customized node-link layouts, as well as the structural editing in matrix representations have not been considered so far.

3. Besides the structure of graphs, attribute data associated with nodes and edges often exist and can be of interest. Although visualization can also support the user in editing such data, the visual editing of the graph's attribute values has yet to be investigated.

4. For developing novel approaches supporting the visual editing of graphs, fundamental tasks which have to be carried out by the user must be determined. On an abstract level, four categories of visual graph editing tasks with different goals can be distinguished: *Analyze Data*, *Select or Specify Data to Edit*, *Apply Edit Operation* and *Adjust Representation* tasks. The processing sequence of tasks of various graph editing scenarios can be described with the *extended general editing model* developed within this thesis.

5. The editing of the graph's structure in node-link representations implies changes of the graph layout. When working with established and customized node-link layouts of larger graphs, a suitable editing solution must allow users to control the final outcome without layouting every element by hand. Moreover, changes to the graph layout must remain local to preserve the user's mental map of the data.

   In this thesis, a novel editing approach based on the concept of lenses was developed which meets these requirements. The semi-automatic *EditLens* enables users to interactively define a local region where an edit operation affects an already customized and established graph layout. Different layout solutions are computed automatically according to existing layout constraints and aesthetic criteria and are then suggested to the user. Eventually, the user may accept a suggested solution and commit the edit operation.

6. Different visual graph representations have different strengths and weaknesses for supporting the user in editing a graph's structure. As presented in this thesis, matrix representations can suitably support the editing of single as well as multiple edges of a graph.

7. For facilitating the editing of attribute values associated with nodes or edges, a visual graph representation communicating graph attributes alongside the graph's structure becomes necessary as a basis. For this purpose, representations with attribute-dependent or attribute-independent layouts can be used.

   On this basis, novel approaches for directly editing graph attribute data have been developed. When using node-link representations with attribute-dependent layouts, the editing of attribute data can be accomplished by interactively changing the graph layout. Exemplary interaction techniques for editing both, quantitative and qualitative node attribute values in scatterplot and semantic substrates layouts were developed within this thesis.

   For directly editing attribute values in node-link representations with attribute-independent layout, different interaction techniques are needed. For this purpose, a novel pop-up editing interface was developed which provides controls for editing qualitative and quantitative values. This interface can be invoked on a node or edge of interest, is integrated into the graph representations and allows the editing in place.

   As changes to local attribute values and the global value distribution must be communicated upon applying edit operations, dedicated visual feedback was developed for both kinds of representations.

8. Working with visual graph representations almost always includes navigating to different parts of the graph. However, when exploring the data for determining subsets to be edited, situations might occur where it is difficult for the user to decide where to go next. To support the user's navigation decisions and to mitigate trial-and-error navigation, a novel approach for providing navigation recommendations to interesting data targets was developed within this thesis. The basic idea is to determine recommendation candidates in the context of a user defined focus, to detect the most interesting candidates based on a degree-of-interest function afterwards, and to eventually communicate navigation recommendations to the most interesting targets visually.

9. Due to the complexity of graph data, situations may occur where users have to either deal with representations visualizing a subset of the data only or representations that are cluttered due to too many information. This may hinder the analysis and editing of details. In this thesis, user support with interactive lens techniques was investigated. Within a comprehensive overview of existing lens techniques, lenses for different data types and different tasks were surveyed. As a result, several lens techniques for adding necessary or suppressing irrelevant details in graph

representations could be identified. To capture the concept behind lenses, key properties of lenses were described and a general model of lenses was established, which describes how lenses are integrated into the visualization process. This conceptual view can facilitate the development of novel lens techniques for supporting visual graph editing.

10. The visual editing of graph data relieves the user from cognitively expensive switches between different working environments for carrying out analysis or editing tasks. This can be beneficial in various situations. Examples include the correction or cleansing of unknown graphs prior to analysis and the analysis of different what-if-scenarios.

11. All approaches developed within this thesis were implemented and applied to different use cases. Moreover, several user studies were conducted to collect user feedback for improvement.